**Московский государственный университет имени М.В. Ломоносова**

**Факультет вычислительной математики и кибернетики**

**Кафедра математических методов прогнозирования**

Толстихин И. О.

# Разработка методов классификации зловредных исполняемых файлов

Отчет о проделанной работе

**Москва**
**2009**

# Содержание

# 1 Введение

В условиях активного развития компьютерных технологий и сети Интернет в жизни пользователей персональных компьютеров всё большую роль начинают играть вопросы информационной безопасности. Это обусловлено активным ростом числа всевозможных вредоносных программ, распространяющихся в сети, а также злоумышленников, стремящихся извлекать выгоду, используя подобного рода программы.

Для обеспечения безопасной работы пользователей с компьютерами разработано большое число антивирусных программ, основной задачей которых является защита системы от любых угроз, способных причинить ей вред.

Большинство современных антивирусов использует так называемый "сигнатурный" метод детектирования, основанный на поиске в файлах предопределённых последовательностей байт. Эти последовательности называются "сигнатурами" и большая часть антивирусных баз состоит именно из них.

Основной проблемой сигнатурного метода детектирования является неспособность обнаруживать новые варианты зловредных программ, поскольку для этого ему требуется наличие соответствующей сигнатуры в базах. Таким образом, при появлении нового вида зловредных программ пользователи остаются незащищенными до следующего обновления их антивирусов. Также стоит отметить, что в большинстве случаев сигнатуры извлекаются "вручную" профессионалами, работающими в антивирусных компаниях.

Последнее время у антивирусных компаний приоритетным направлением является так называемое эвристическое детектирование файлов. Этот подход основан на использовании статистической информации для поиска зловредных программ. В данной работе речь пойдёт именно об этом подходе, способном, в отличие от сигнатурного метода, обнаруживать вредоносный функционал в новых вариантах зловредных программ.

# 2 Детектирование зловредных файлов

В данной главе будет подробно рассмотрена задача детектирования зловредных файлов. Будут приведены все определения, необходимые для понимания дальнейшего материала. Также будут подробно рассмотрены современные подходы к решению задачи детектирования зловредных файлов. В конце главы будет дана формальная постановка задачи, решаемой в данной работе.

## 2.1 Определения

В этом разделе мы достаточно подробно рассмотрим понятия, связанные с информационной безопасностью. Здесь будет приведена информация, необходимая для обсуждения основной темы данной работы – детектирования зловредных файлов.

### 2.1.1 Информационная безопасность

Понятие *информационная безопасность* (information security) представляется интуитивно ясным. Тем не менее мы формализуем его во избежание путаницы в дальнейшем. Мы будем придерживаться следующего определения. Под информационной

безопасностью понимается "защищенность информации и поддерживающей ее инфраструктуры от любых случайных или злонамеренных воздействий, результатом которых может явиться нанесение ущерба самой информации, ее владельцам или поддерживающей инфраструктуре" (ЗАО "Лаборатория Касперского", 2009).

В данной работе под *воздействием* будем понимать присутствие в системе запущенного исполняемого файла, способного причинить ей вред. В связи с этим, нас будет интересовать следующий вопрос, изучаемый в рамках информационной безопасности, – обнаружение таких файлов.

### 2.1.2 Зловредные файлы

Предметом нашего изучения будут зловредные исполняемые файлы, более известные под термином *malware* (сокращение от Malicious Software). Далее мы будем иногда опускать слово "исполняемый", поскольку речь будет идти именно о компьютерных программах.

Зловредным считается любой исполняемый файл, выполняющий вредоносные действия, включая:

- подвержение риску безопасности системы;
- повреждение системы;
- получение доступа к личной информации владельца системы без его ведома.

Отметим важный факт – в рамках данной работы изучаются исключительно PE файлы (Portable Executive) – формат исполняемых файлов, используемый в 32-х и 64-х битных версиях операционной системы Microsoft Windows. О свойствах этого формата будет рассказано в следующем разделе.

### 2.1.3 PE формат и известные команды

Portable Execurive – это, в некотором смысле, "родной" формат исполняемых файлов в 32-х разрядной архитектуре MS Windows (Win32). Слово "переносимый" (portable) означает, что файловый формат универсален для платформы Win32: PE загрузчик любой Win32 платформы, за редким исключением, сможет распознать и использовать этот формат. Каждый исполняемый файл, предназначенный для платформы Win32, использует PE формат. Вот почему наше изучение зловредных файлов начнётся именно с рассмотрения формата PE.

PE формат представляет собой стрктуру, содержащую в себе всю необходимую при исполнении файла информацию для загрузчика ОС Windows. Важым является *недокументированность* формата PE. Поэтому все знания о нём основаны на опыте специалистов и профессионалов, а также на большом числе статей и обзоров, написанных ими.

PE формат имеет достаточно сложное устройство, и на его изучение может уйти много времени. В этой главе будут рассказаны только самые основные моменты, необходимые нам для дальнейшего обсуждения темы.

**Структура PE Файла.** Основой структуры PE формата является так называемый PE заголовок (`PEHeader`). Он находится по определённому смещению от начала файла, которое указано в двойном слове по адресу `0x3C`. (Здесь и всюду далее будет использоваться 16-ричное представление чисел, поскольку именно оно является общепринятым при обсуждении исполняемых файлов.) В PE заголовке присутствует вся основная информация как о свойствах исполняемого файла на этапе выполнения, так и о размещении структур, входящих в состав PE формата, внутри файла.

Для того, чтобы читатель имел некоторое представление об исполнении файла в ОС Windows, опишем в нескольких словах этот процесс. При запуске приложения (исполняемого файла) система выделяет область в оперативной памяти, необходимую для загрузки файла. Затем, используя различную информацию из PE заголовка, происходит непосредственно загрузка файла в память с его проектированием на виртуальное адресное пространство. После ещё несколько событий управление наконец передаётся на `точку входа` (Entry Point, EP) файла.

Отсюда возникает несколько основополагающих понятий, связанных с адрессацией, – `RVA` (Relative Virtual Adress), `VA` (Virtual Adress) и `ImageBase`. `ImageBase` – это виртуальный адрес в памяти, начиная с которого происходит загрузка файла. `VA` – виртуальный адрес. `RVA` – относительный виртуаьный адрес:

$$RVA = VA - ImageBase.$$

Итак, PE заголовок содержит, помимо всего прочего, всю необходимую информацию о размещении файла в виртуальном адресном пространстве.

После PE заголовка последовательно идут структуры, описывающие так называемые *секции* PE файла. В этих структурах содержится информация о физическом и виртуальном размере каждой секции, о её смещении относительно начала файла, а также о её `RVA` (*ещё раз поясним для понимания – под RVA секции понимается её виртуальное смещение относительно ImageBase. Оно может не совпадать со смещением в файле из-за выравнивания секций. Углубляться в эти вопросы мы не будем.*)

Далее в определённом порядке идут сами секции.

Всё, что расположено в файле за последней секцией при загрузки файла игнорируется. Эту часть файла принято называть *overlay*.

Так, вкратце, выглядит структура PE формата.

**Экспорт и импорт исполняемого файла.** Важными понятиями явдяются экспорт и импорт исполняемого файла. В PE заголовке есть `RVA` на так называемые *таблицу импорта* и *таблицу экспорта*.

В таблице импорта присутствует информация о *внешних* функциях, необходимых исполняемому файлу для работы. При загрузке приложения в память происходит автоматическое заполнение этой таблицы действительными виртуальными адресами перечисленных функций. Эти адреса указывают на динамические библиотеки (Dynamic link library, dll) внутри которых содержатся запрашиваемые функции. Говорят, что файл *импортирует* эти функции. Необходимые библиотеки заранее загруженные в память перед запуском приложения или подгружаются в память в процессе работы приложения. При этом говорят, что библиотека *экспортирует* функции. Динамические библиотеки – такие же исполняемые PE файлы, как, например, и `exe`-файлы.

Соответственно, в таблице экспорта присутствует описание экспортируемых данным файлом функций, к которым могут обращаться другие приложения.

**WinAPI – Windows Application programming interface.** В операционной системе Windows существует набор системных функций, которые дают возможность программистам обращаться к системе и взаимодействовать с ней. Этот набор функций и называется WinAPI (в дальнейшем – просто API).

WinApi функции экспортируются основными системными библиотеками, такими как `kernel32.dll, user32.dll` и так далее. При работе приложения, использующего WinAPI функции, оно обращается к системным библиотекам, также загруженным в память.

Среди API функций выделяются так называемые `Native Windows API`. Они являются самыми основными функциями, экспортируемыми библиотекой `ntdll.dll`. Эти функции делятся на два типа – на системные вызовы ядра, передающие управление функциям ядра из `ntoskernel.exe`, и функции, реализованные в пользовательском режиме. Большая часть этих API функций может вызываться из любого приложения, в том числе из *драйверов*. (Драйвер - исполняемый PE файл режима ядра. Чаще всего драйверы предназначены для предоставления другим программам доступа к тому или иному аппаратному обеспечению некоторого устройства. Например, есть драйверы файловой системы, драйверы клавиатур и так далее.)

Также отметим, что `Native WinAPI` в большинстве своём недокументированы или слабодокументированы.

**Известные команды.** Как известно, "родным" языком Win32 приложений является ассемблер. Нам понадобятся знания о некоторых важных командах ассемблера, которые будут описаны в этом параграфе. Также здесь будет приведено описание нескольких важных API функций.

`jmp`. Это команда безусловного перехода на адрес, который она получает в качестве своего аргумента.

`call`. Данная команда – одна из основных при работе приложения. С её помощью выполняются вызовы процедур, описынных программистом, а также вызовы API и других функций. В качестве аргумента команда `call` принимает, как правило, `RVA` адреса, на который необходимо передать управление. Её отличие от команды `jmp` заключается в том, что перед передачей управления на обозначенный адрес происходит запись в стек `VA` возврата – или, что то же самое, `VA` следующей за `call` операцией. Вызываемая процедура или функция, как правило, заканчивается операцией `retn`.

`retn`. Эта операция извлекает из стека адрес возврата и возвращает на него управление.

Нам также понадобятся некоторые арифметические операции.

`xor`. Побитовое логическое сложение по модулю 2 аргументов.

`and`. Побитовое логическое `И`.

`or`. Побитовое логическое `ИЛИ`.

И, наконец, опишем работу двух API функций.

`LoadLibrary`. Этот системный вызов загружает в память библиотеку с указанным в качестве аргумента именем и возвращает её виртуальный адрес. В случае неудачи возвращает ошибку.

`GetProcAddres`. Эта функция принимает в качестве аргументов адрес загруженного приложения и имя функции. В случае, если заданное приложение экспортирует функцию с заданным именем, API возвращает адрес этой функции. В противном случае – ошибку.

## 2.2 Классификация зловредных файлов

Для понимания устройства зловредных программ рассмотрим немного подробнее их функционалы. В зависимости от своего поведения они делятся на три основных класса - *трояны* (Trojan), *вирусы* (Virus) и *черви*(Worm). Далее всюду будем использовать именование семейств и классов зловредных программ, принятое в компании ЗАО "Лаборатория Касперского".

### 2.2.1 Трояны

Это резидентные зловредные исполняемые файлы, функционирующие в пределах системы. Попав в систему, они укрепляются в ней и начинают выполнять своё предназначение.

**Функционал.** Чаще всего они предназначены для кражи разнообразных паролей, предоставления злоумышленникам удалённого доступа к заражённым системам и для загрузки других зловредных файлов из Интернета.

**Распространение.** Этот вид зловредных программ самостоятельно не может распространяться. Чаще всего трояны попадают в систему с помощью червей, о которых будет рассказано ниже, или загружаются на компьютер другими троянами.

**Виды троянов.** В зависимости от специфики действий происходит деление троянов на подклассы - `Trojan-Spy` (шпионские программы), `Trojan-PSW` (программы, ворующие пароли), `Trojan-Downloader` (программы, загружающие из Интернета другие зловредные файлы).

Отдельно стоит отметить подкласс троянов Backdoor. Это трояны, позволяющие злоумышленнику получить полный или частичный контроль над заражённой системой с помощью удалённого доступа. Backdoor считаются наиболее опасным видом Троянов. Помимо несанкционированного доступа к системе, заражённой данным видом трояна, существует ещё одно применение этих зловредных файлов. Представим, что злоумышленнику удалось заразить большое число компьютеров. (А число может быть действительно огромным. Например, недавно нашумевший сетевой червь `Net-Worm.Win32.Kido` заразил порядка десяти миллионов компьютеров). Это означает, что он располагает гигантскими вычислительными ресурсами. Сеть заражённых компьютеров принято называть *ботнетом* (botnet) или *зомби-сетью*. С помощью ботнетов обычно осуществляются атаки сайтов и сервисов, конечная цель которых – шантаж владельцев атакуемого ресурса.

Также существует не менее опасный подкласс Rootkit (руткит). В него входят, как правило, драйверы, глубоко внедряющиеся в систему. Благодаря использованию функций ядерного уровня, а также доступу к системным структурам, этот вид зловредов особенно сложно обнаружить. Обычно руткиты предназначены для скрытия файлов или процессов, присутствующих в системе. Зловредный драйвер способен перехватывать все системные вызовы и перенаправлять их на свой код.

### 2.2.2 Черви

Черви – зловредные исполняемые файлы, использующие для распространения сеть.

**Функционал.** Определённого назначения у червей нет. Они могут переносить с собой троянов, так же как трояны воровать пароли и предоставлять удалённый доступ к системе.

**Распространение.** Для своего распространения черви используют компьютерные сети.

**Виды червей.** В зависимости от способов размножения черви делятся на `Email-Worm` (распространение с помощью email), `P2P-Worm` (распространение с помощью p2p сетей), `IM-Worm` (с помощью ICQ и других IM клиентов), `IRC-Worm` (с помощью IRC, MIRC и прочих каналов) и `Net-Worm`(с помощью *уязвимостей* системы). Уязвимость – та или иная недоработка системы, которую можно использовать для выполнения несанкционированных действий.

Самым опасным видом червей являются сетевые черви `Net-Worm`. Они распространяются, используя "дырки" в системах. Среди известных представителей этого подкласса червей можно назвать `Net-Worm.Win32.Slammer`, распространявшегося с помощью ошибки в программном обеспечении MS SQL и, фактически, на время расколовшего Интернет вследствие активного размножения. `Net-Worm.Win32.Kido`, упомянутый выше, также является представителем подкласса сетевых червей.

### 2.2.3 Вирусы.

Вирусы – зловредные программы, заражающие файлы, находящиеся в системе. В данном случае речь идёт о PE вирусах – вирусах, заражающих исполняемые файлы.

**Функционал.** Как и в случае червей функционал может быть разным.

**Распространение.** Вирусы распространяются путём заражения других исполняемых файлов. Попав в систему, вирус ищет подходящие ему файлы. Затем он их заражает. Заражение происходит по-разному, в зависимости от вида вируса. При исполнении ни о чём не подозревающим пользователем заражённого файла вместе с оригинальным кодом файла исполняется и код вируса. Таким образом, пока пользователь играет, например, в заражённую косынку, код вируса заражает следующие файлы.

Такой вид вирусов называется *инфекторами* (infector). Существуют также вирусы-*компаньоны*, копирующие себя в папки под именем существующих в этих

папках файлов. Оригинальные файлы вирусы переименовывают и скрывают. При запуске вируса-компаньона вначале происходит исполнение зловредного файла. Затем вирус запускает переименованный им файл. Таким образом пользователь даже и не подозревает о заражении системы.

Вирусы – одни из самых сложно обнаруживаемых зловредных программ. Поскольку вирусы заражают помимо всего прочего и системные файлы, удалять заражённые файлы нельзя – их надо лечить. Обнаружить заражённый файл крайне тяжело, поскольку в большинстве случаев код вируса мал по сравнению с кодом заражённой программы.

## 2.3 Классические методы детектирования

Современные антивирусные компании используют в основном одинаковый набор средств, позволяющих им обнаружить зловредные файлы в системе. Если антивирус признаёт конкретный файл зловредным, то говорят, что он *детектирует* его. Детектирование зловредных исполняемых файлов является основной задачей антивирусов. В этой главе вкратце перечислены основные методы детектирования зловредных файлов, применяемые сегодня антивирусами.

### 2.3.1 Сканирование файла

Это основной метод, на котором строятся все современные антивирусы. Сканирование файла – поиск в нём заранее известной последовательности байт. Эта последовательность называется *сигнатурой*, поэтому данный метод также известен под названием "сигнатурного детектирования". Антивирусные базы современных продуктов большей частью состоят именно из сигнатур зловредных файлов.

Сигнатуры извлекаются специалистами, разрабатывающими антивирус. Когда к ним в руки попадает зловредный файл, они анализируют его функционал, определяют, к какому классу и подклассу относится данный зловредный файл, и выделяют из него уникальную сигнатуру. Затем сигнатура добавляется в базы.

### 2.3.2 Отслеживание активности

Это направление всё больше и больше развивается в последнее время. Оно основано на отслеживании деятельности исполняемого файла в рамках системы. Отслеживание может происходить различными методами – с помощью драйверов, перехватывающих системне вызовы, с помощью мониторов, фиксирующих изменения файлов или настроек системы. В полученных таким образом лог-файлах автоматически происходит поиск подозрительной деятельности.

Для реализации этого подхода исполняемый файл необходимо запустить. Поэтому такие подходы к детектированию файла, как отслеживание активности, называются *поведенческим* анализом файла. В отличие от отслеживания сканирование файла является *статическим* анализом файла.

Как правило, изучаемый файл запускается в искусственно обустроенной среде – на виртуальной машине или на специально выделенной для этих целей системе. В этой системе заранее производится множество настроек, которые помогут отслеживать поведение запускаемого файла. Такие системы принято называть *песочницами* (sandbox).

Понятно, что этот метод детектирования необычайно ресурсоёмок. Для анализа одного файла его необходимо запустить и снять показатели мониторов. Только после этого начинает работать система детектирования.

### 2.3.3 Эвристическое детектирование

Эвристичекие методы детектирования – методы, основанные на статистической информации о виде, свойствах и поведении зловредных программ. Они могут быть как поведенческими, так и статическими. Это направление сегодня является наиболее приоритетным для многих антивирусных компаний. Основным преимуществом данного подхода является возможность детектировать ранее не встречавшиеся зловредные файлы.

Для сигнатурного детектирования необходимы сигнатуры, извлекаемые из **уже имеющихся** вариантов зловредных программ. Часто одна сигнатура способна срабатывать на целом семействе зловредных файлов с похожей структурой. Однако, если свет увидит новый тип зловредных файлов, имеющих структуру, отличную от всех предыдущих, сигнатурное детектирование будет бессильно против него.

Именно для таких случаев разрабатывается концепция эвристического детектирования. В данной работе речь пойдёт именно об эвристическом детектировании зловредных файлов.

## 2.4 Недостатки существующих методов детектирования

У всех существующих сегодня методов детектирования есть свои недостатки.

Метод детектирования интересует нас в данном случае именно с точки зрения его применимости в антивирусе, следовательно нам важны следующие характеристики метода:

- скорость работы;
- способность обнаруживать и обезвреживать зловредные файлы;
- низкий уровень *ложных срабатыний.*

Ложное срабатывание – ситуация, когда антивирус детектирует чистый файл. Это чрезычайно опасное явление. Во-первых, если антивирус детектирует, например, чистый системный файл и удаляет его из системы, это может привести к самым серьёзным последствиям, вплоть до отказа функционирования системы. Во-вторых, детектирование чистых файлов сильно подкашивает авторитет антивируса, а значит наносит удар по его бюджету.

Учитывая всё вышесказанное, часть из перечисленных методов автоматически выпадает из рассмотрения.

В первую очередь мы отбрасываем все поведенческие методы детектирования, поскольку они чрезвычайно медленны и ресурсоёмки. Нас интересует только статический анализ файлов. Сигнатурное детектирование – оправдавший себя классический метод. Но и в нём есть уже названный недостаток – он не способен детектировать новые виды зловредных файлов, подвергая систему опасности до обновления антивирусных баз.

Хорошей иллюстрацией к вышесказанному является эксперимент, описанный в [1], в рамках которого продукты основных антивирусных компаний испытывались на способность детектировать обфусцированные файлы.

*Обфускация* (запутывание кода) – намеренное приведение исполняемого кода файла к виду, сохраняющему его функционал, но затрудняющему анализ и понимание алгоритма работы программы.

Авторами статьи были обфусцированы несколько известных зловредных файлов, детектировавшихся тремя рассмотренными антивирусами. В процессе обфускации применялось несколько простейших приёмов запутывания кода. Среди прочих использовался метод, известный под названием `dead-code insertion` (вставка в код оригинального файла незначащих команд, например – последовательности интсрукций `inc eax; dec eax`). Также авторы применяли более сложную технику обфускации – *трансформацию кода* (code transposition). При трансформации кода исполняемый код разрывается на части, которые перемешиваются между собой. Исходный функционал сохраняется за счёт вставки команд `jmp`, осуществляющих переходы на нужные части кода.

Обфусцированные файлы подавались на вход антивирусов для проверки детектирования этих файлов. Результат оказался удивительным – ни один из антивирусов не справился с задачей. Ни один из них не смог распознать в обфусцированных элементарным образом файлах уже известные ему зловредные программы. Стоит иметь ввиду, что опытные писатели зловредных программ применяют куда более хитрые методы обфускации, чем простую вставку незначащих команд.

Таким образом, единственный метод, интересующий нас – эвристическое детектирование, которое в большинстве случаев использует известные алгоритмы теории машинного обучения.

## 2.5 Постановка задачи

Мы пришли к формальной постановке интересующей нас задачи.

Задачу детектирования предлагается рассматривать как обыкновенную задачу **бинарной классификации**. Множество **классов** представлено зловредными программами и чистыми программами. Классифицируемые **объекты** – PE файлы.

Первой же проблемой, с которой мы сталкиваемся при решении обозначенной задачи классификации, является **отсутствие признакового описания** у классифицируемых объектов. Нам бы хотелось научиться извлекать из PE файлов признаки. Причём извлекать их статически, то есть не запуская на исполнение рассматриваемый файл.

> Основной задачей настоящей работы является статическое извлечение из исполняемых файлов признаков, на основе которых в будущем будет решаться первоначальная задача бинарной классификаци.

# 3 Известные результаты

В этой главе мы рассмотрим известные результаты, полученные в области использования алгоритмов машинного обучения при детектировании зловредных файлов. Как было сказано, нас интересует только статистический анализ файла. По этой причине мы не будем рассматривать многие работы, в которых авторы получили

интересные результаты на основе поведенческого анализа файлов.

## 3.1 Подходы

Несмотря на то, что основой современных антивирусов является сигнатурное детектирование, исследования на тему эвристического анализа файлов ведутся уже довольно давно. В некоторых работах получены неплохие результаты. Среди них будет рассмотрено три работы, которые представляются самыми интересными.

### 3.1.1 Подход "обфускация - диобфускация"

Один из самых интересных подходов к статическому анализу исполняемых файлов представлен в [1]. Авторы рассматривают задачу детектирования файла как "игру в обфускацию-деобфускацию". Имеется ввиду, что происходит постоянная гонка между антивирусными компаниями, детектирующими новые модификации зловредных файлов, и злоумышленниками, постоянно пытающимися обфусцировать функционал своих зловредных файлов.

Таким образом, основная задача антивируса – диобфусцировать запутанный код и получить оригинальную последовательность команд для получения представления о работе файла. Диобфускация при этом – задача, обратная к обфускации.

В ходе работы авторы в основном работали с *полиморфными* вирусами – вирусами, обфусцирующими свой код при каджом последующем заражении.

Для решения этой задачи авторы предлагают использовать так называемые *графы управления* или, иначе, *граф потока управления* (Control Flow Graph). Это достаточно известное понятие, поэтому опишем его лишь в нескольких словах. Граф управления – это структурная модель программы, способная показывать связь между её элементами. Вершинами графа управления являются операторы ветвления, встречающиеся в коде программы, и части кода, где эти ветвления сливаются. Дуги – операторы обработки и передачи информации.

Алгоритм дететкирования, предложенный авторами, состоит из двух этапов. Вначале на основе изучаемого файла строится его граф управления. Затем начинает работать обфускатор, созданный авторами. Его задача – на основе графа управления сгенерировать некоторое число производных графов, не отличающихся по функционалу от оригинального. По сути, на этом этапе происходит получение графов управления обфусцированных версий изучаемого файла. Среди полученных таким образом графов уже ищутся признаки, сгенерированные на основе имеющихся зловредных файлов.

В ходе экспериментов алгоритм продемонстрировал безошибочное детектирование всех обфусцированных версий вируса. Также он прошёл все испытания с чистыми файлами.

Главная проблема данного подхода – большое количество времени, занимаемой процессом генерации графа и последующей его обфускации. Время работы их алгоритма с файлом размера 1 Мегабайт достигало 800 секунд. Очевидно, что такой алгоритм не может быть пока что применён в составе антивирусов.

### 3.1.2 Использование алгоритмов машинного обучения для детектирования зловредных файлов

Во многих работах были предприняты попытки решать задачу классификации зловредных файлов, применяя классические алгоритмы теории машинного обучения. В большинстве своём все они основаны на классической статье [3], поэтому мы ограничимся рассмотрением результатов, приведённых в этой работе.

Авторы статьи проделали достаточно большую работу, применив 3 известных алгоритма с использованием 3 разных видов признаков.

Были использованы следующие признаки:

- Имена API и библиотек, используемых исполняемым файлом;
- Последовательности байт, встречавшихся в файле;
- Строки, входящие в состав файла.

На основе данных признаков происходила настройка трёх алгоритмов. В таблице перечислены алгоритмы и признаки, с которыми эти алгоритмы настраивались.

| | |
|---|---|
| Наивный Байес | Строки |
| RIPPER | API и библиотеки |
| Композиция наивных байесовских классификаторов | Байты |

Настройка алгоритмов происходила с помощью выборки, состоящей из 1001 чистого файла и 3265 зловредных файлов. Подробной информации о разделении выборки на контрольные и обучающие подвыборки приводить не будем. Приведём лишь результаты, полученные авторами.

В таблице использованы стандартные обозначения: TP – число правильно классифицированных зловредных файлов, FP – число неправильно классифицированных чистых файлов, TN – число правильно классифицированных чистых файлов, FN – число неправильно классифицированных зловредных файлов.

| Алгоритм | TP | TN | FP | FN |
|---|---|---|---|---|
| Наивный Байес | 3176 | 960 | 41 | 89 |
| RIPPER | 20 | 195 | 11 | 18 |
| Композиция наивных байесовских классификаторов | 3191 | 940 | 61 | 74 |

Данные результаты были получены авторами при проверке работы алгоритмов на новой выборке такого же размера, как и первая. При этом авторы никак не комментируют результаты алгоритма RIPPER. Их происхождение остаётся непонятным, поскольку сумма приведённых величин должна дать размер всей выборки. Будем считать, что RIPPER тестировался на выборке размером 244 файла.

Как мы видим, лучший результат показал наивный байесовский классификатор, обученный по строкам. Авторы также приводят некоторые из правил, сгенерированных при настройке алгоритмов. Очень интересным является тот факт, что RIPPER в качестве одного из правил получил следующее: *если в импорте файла присутствуют функции библиотеки* `msvbvm`*, то он зловредный*. Это означает, что все файлы, написанные на языке Visual Basic детектировались алгоритмом RIPPER,

что является весьма смелым шагом, поскольку многие известные чистые приложения реализованы именно на этом языке. Можно сделать вывод, что в выборке было небольшое число файлов, написанных на языке VB, и все они были зловредными.

Также веьма спорными являются и результаты настройки наивного байесовского классификатора. Правила, получаемые с помощью него, – это вероятность принадлежности файла классу, при наличии в нём признака. Одним из правил, полученных в ходе настройки, является следующее: если в файле есть строка "windows", то с вероятностью 45/47 файл чистый. С таким правилом никак нельзя согласиться. Большое число зловредных файлов использует именно такого рода строки для маскировки. Также зловредные файлы достаточно часто используют имена ключей реестра. Одной из основных ветвей реестра является `HKLM\SOFTWARE\MICROSOFT\WINDOWS`, в имя которого, как мы видим, входит строка "windows". Ещё одним весомым аргументом является тот факт, что львиная доля зловредных файлов в процессе своей работы обращается к системной папке – `WINDOWS\SYSTEM32`.

## 3.2 Недостатки подходов

Бесспорно, существуют алгоритмы с чрезвычайно высоким уровнем точности. Это означает, что они способны детектировать новые версии вирусов и при этом обходиться без ложных срабатываний. Таким алгоритмом, например, является рассмотренный алгоритм построения графа управления. Как правило, основной проблемой таких алгоритмов неприемлимо долгое время их работы.

Проблема подходов, основанных на классических алгоритмах теории машинного обучения, состоит в неподходящих признаках, извлекаемых авторами. Алгоритмы, построенные на этих признаках, можно легко обойти, используя шифрование строк, скрытое получение адресов API и обфускацией кода. Про эти приёмы будет подробнее рассказано ниже.

# 4 Использование эвристик при извлечении признаков

Во всех прошлых исследованиях вопроса классификации зловредных файлов, как правило, авторы использовали признаки, лежащие на поверхности PE формата. Например, в упомянутых статьях использовались такие признаки, как строки, имена API функций, а также последовательности байт, встречающиеся в файле. Все эти признаки, конечно, способны кое-что сказать о свойствах, а реже – и о функционале файла. Однако, гораздо чаще для изучения файла необходимы специализированные признаки, способные выделить в нём более глубокие и осмысленные свойства, чем наличие строк и нескольких байт.

Главной идеей, лежащей в основе данной работы, стала попытка применить знания об общем устройстве и специфике зловредных файлов при извлечении признаков. Результатом должны стать признаки, с одной стороны, покрывающие достаточно много файлов, а с другой - характеризующие целые группы чистых или же зловредных файлов и способные сказать хоть что-то о функционале или устройстве этих файлов.

В ходе работы было принято решение извлекать признаки 3-х видов:

- Признаки, описывающие свойства PE формата.
- Признаки, описывающие свойства файла, как простой последовательности байт.
- Эвристические признаки - попытка обобщения знаний о зловредных файлах в рамках признаков.

При этом в данной работе рассматриваются только бинарные признаки.

## 4.1 Свойства PE формата

Это признаки, опирающиеся на свойствах PE формата. Они призваны дать нам представление о структуре исполняемого файла. К ним относится всё, что касается информации, заключённой в PE заголовке, а также всего, на что он ссылается.

Среди всех прочих предлагается рассматривать только следующие признаки: тип исполняемого файла, импортируемые API функции, экспортируемые API функции.

**Тип файла.** При изучении свойств и функционала исполняемых файлов важным является их тип. Они могут быть трёх видов – **.exe** файлы, динамические библиотеки (**.dll**) и драйверы (**.sys**).

Всвязи с этим возникает **три бинарных признака** – по одному на каждый из трёх типов файлов. Признак равен единице, если рассматриваемый файл имеет тип, которому соответствует данный признак. Эти признаки в некоторых случаях достаточно информативны в сочетании с другими, дополняющими их признаками.

Информацию о том, с каким из трёх видов мы имеем дело, легко **получить из PE заголовка**. Она представлена в нём в виде флагов, находящихся в известных местах.

Приведём пример. Допустим, мы имеем зловредный драйвер – руткит. Достаточно большое число руткитов имеют одинаковую схему работы. В операционной системе Windows существует недокументированная структура – `KeServiceDescriptorTable`. Она состоит из полей, каждое из которых содержит номер сервиса и адрес. При вызове драйвером какой-либо Native WinApi Функции (функции режима ядра) система в конечном счёте сопоставляет номер вызываемого сервиса с адресом в `KeServiceDescriptorTable` и передаёт выполнение по соответствующему адресу.

Драйверы могут импортировать эту структуру для различных целей - как легальных, так и злоумышленных. Например, драйверы большинства антивирусов перехватывают вызовы API функций для обнаружения подозрительной активности в системе. Это делается путём подмены в `KeServiceDescriptorTable` соответствующих адресов.

Точно также руткит может подменить адреса сервисов своим кодом, что позволит ему получить управление во время вызова того или иного сервиса. Таким образом, драйвер, загруженный в систему, может скрывать от юзера файлы или процессы.

Подытоживая сказанное, можно сделать вывод – если перед нами драйвер, импортирующий `KeServiceDescriptorTable`, то высока вероятность, что он зловредный.

**Импорт PE файла.** В самом общем случае, импорт исполняемого файла – это те API функции, которыми он будет пользоваться при работе.

Импортируемые API функции **перечисляются в специальной** *таблице импорта*, **находящейся по адресу, указанному в PE заголовке**. Таким образом, иногда удаётся совершенно спокойно получить список импортируемых функций, а также имена динамических библиотек, которые экспортируют эти функции.

Интуитивно ясно, что импортируемые исполняемым файлом функции дают если и не исчерпывающую, то по крайней мере достаточную для анализа поведения файла информацию. В данном случае, слово "импортируемые" употреблено в значении "подгружаемые при работе". Здесь встаёт вопрос – совпадает ли множество импортируемых функций с множеством функций, перечисленных в известной нам таблице импорта?

Ответ, к сожалению, в большинстве случаев отрицательный. Если чистые файлы объявляют весь свой импорт в таблице импорта, то зловредные файлы склонны скрывать какую-либо информацию о своём функционале и импортировать функции скрытыми путями. В частности, они могут использовать такие API функции, как `LoadLibrary` и `GetProcAdress`, а имена API и библиотек, передаваемые этим функциям, хранить в закриптованном виде. В этом случае всё, что им нужно – импортировать две вышеупомянутые функции. Часто дела обстоят ещё хуже – дело в том, что существует много "обходных" способов получения адреса загруженной в память `kernel32.dll`. Самый популярный вариант – через структуру TEB, в которой есть указатель на структуру PEB, в которой, в свою очередь, имеется информация о адресах загруженных модулей. Получив адрес библиотеки, можно спокойно найти адрес любой экспортируемой ею функции, всего лишь используя PE заголовок. Таким образом, возможны случаи, что в таблице импорта вообще пусто или API функции, перечисленные в ней, не имеют отношения к настоящему функционалу файла.

По этим причинам было принято решение в качестве импорта использовать не только функции, перечисленные в таблице импорта файла, но и все имена функций, встречающиеся в файле. Такие имена функций извлекаются из файла на этапе поиска строк, который будет описан ниже.

Для отображения импорта в качестве признаков было принято решение составить некоторый список API функций и отмечать присутствие в файле функций из списка. Список предлагается генерировать из самых часто встречаемых среди зловредных и чистых файлов функций. Само множество функций, из которого выбирается список, – экспорты основных библиотек ОС Windows, таких как `kernel32.dll, shell32.dll, advapi32.dll, gdi32.dll, user32.dll`, и отдельно для драйверов - экспорты `ntoskrnl.exe`.

Таким образом, мы имеем число признаков, равное числу функций в полученном нами списке. Равенство единице одного из признаков означает, что в файле присутствует API функция, соответствующая ему.

**Экспорт PE файла.** По аналогии с таблицей импорта, существует *таблица экспорта*, в которой перечислены экспортируемые файлом функции.

Чаще всего экспорт имеют динамические библиотеки. При этом на практике редко встречаются случаи, когда зловредные библиотеки используют говорящие имена экспортируемых функций или маскируются, используя имена экспорта чистых библиотек. Чаще всего функции зловредных библиотек вовсе не именуются и обращение

к ним происходит по так называемым "ординалам" – своего рода нумерации функций.

Суммируя всё вышесказанное было принято решение в качестве признака взять лишь **наличие функций в таблице экспорта** и никак не учитывать информацию про имена этих функций.

## 4.2 Свойства файла

До сих пор мы рассматривали исполняемый файл с точки зрения структуры, продиктованной PE форматом. Но существует и другой подход. Мы можем относиться к исполняемому файлу как к простой последовательности байт фиксированной длины.

Признаки из этой категории являются попыткой описать простейшие свойства файла, как упорядоченного массива байт. Сейчас мы будем рассматривать всего два из них – строки, входящие в состав файла, и размер файла.

**Строки.** Строки, встречающиеся в файле,– один из наиболее часто используемых признаков при решении задачи классификации исполняемых файлов. Этот факт вполне оправдан. Во-первых, в число строк входят все имена API функций и динамических библиотек, используемых файлом. А значит, строки дают нам представление о поведении файла в системе. Во-вторых, многие API функции в качестве своих аргументов используют указатели на имена искомых сервисов, создаваемых процессов, удаляемых файлов. Вся эта информация способна уточнить и дополнить общую картину предназначения файла.

Несмотря на то, что строки, как и имена функций, часто хранятся в закриптованном виде, чаще всего именно строки становятся одной из основных "подсказок" при анализе файла.

Приведём несколько примеров, показывающих пользу строк.

Предположим, что перед нами Trojan-Downloader. Чаще всего эти троянцы используют API функцию `URLDownloadToFile`, входящую в состав `utlmon.dll`, для скачивания других зловредных файлов из интернета. Среди параметров этой функции основными являются два указателя – указатель на имя сохраняемого файла и указатель на строку, содержащую адрес скачиваемого файла. В самом простом случае обе этих строки будут в явном виде присутствовать в теле файла.

Таким образом, увидев среди строк имена упомянутой API функции и библиотеки, а также интернет адрес файла, мы смело можем сделать вывод, что в процессе работы этот файл будет загружен на компьютер. Если он окажется зловредным, то разбираемый нами файл – Trojan-Downloader. Конечно, загруженный фал может оказаться чистым, например – обновление ОС Windows. Но главное то, что наличие определённой комбинации строк указало нам на поведение файла, в данном случае – на то, что он загружает из сети другие файлы.

Если бы мы нашли в файле также имя API функции `WinExec`, запускающей приложение, и, например, строку `SeDebugPrivilege`, используемую для повышения прав запущенного приложения, мы бы могли с большой вероятностью утверждать, что наш файл – зловредный.

Для отображения информации, связанной со строками, использовался такой же метод, как и в случае с импортом.

**Извлечение** самих строк из файла происходит простым линейным проходом файла и поиском в нём отображаемых байт. Байт считается отображаемым, если

он входит в предопределённый набор символов, знаков и цифр. Найденные буквы конкатинируются в строки, которые и извлекаются из файла.

Скажем несколько слов и соображений по поводу зашифрованных строк.

Как упоминалось, зловредные файлы в большинстве случаев для скрытия своего функционала хранят строки в зашифрованном виде. Чаще всего для шифровки используются простейшие бинарные операции `XOR`, `AND`, `NOT` или `OR` и их комбинации. В случае `XOR`, как правило, выбирается конкретное однобайтовое значение, называемое *ключом*, и к каждому байту слова последовательно применяется операция бинарного сложения с этим ключом. Таким образом реализуется простейший вариант шифрования. Очевидно, что описанная операция обратима. Достаточно повторно применить её к зашифрованной строке с тем же ключом – и мы получим оригинальную строку.

Отсюда возникает идея, полезная в процессе извлечения строк. При поиске строк в файле можно пытаться применять операцию `XOR` к последовательностям байт с определённым набором ключей. Если файл хранит в себе зашифрованные рассматриваемым способом строки, то при полном переборе ключей нам удастся извлечь оригинальные строки из файла. Проблемой является высокая сложность задачи перебора ключей.

**Размер файла.** Это, наверное, первый признак, который приходит в голову при рассмотрении любого файла. Как ни странно, при изучении исполняемых файлов этот признак оказывается достаточно полезным. Это связано с несколькими причинами.

У каждого языка программирования есть свой компилятор, а у каждого компилятора – свои особености компиляции кода в бинарный вид. Исполняемые файлы, написанные на разных языках, имеют разную структуру. Некоторые из них сильно выделяются среди прочих. Например, значительной частью нынешнего семейства `Trojan-Downloader` являются так называемые `Banload`, которые загружают на компьютер зловредные программы, предназначенные для кражы паролей и информации о банковских картах. Примечательно это семейство тем, что абсолютно все его представители написаны на `Delphi`. При этом из-за особенностей компилятора средним размером такого зловредного файла является $10 - 15$ Мегабайт. Этот факт выделяет среди множества размеров файлов "подозрительные" области.

Ещё одна причина – присутствие среди исполняемых файлов различного рода установочных пакетов. Установочный пакет – самораспаковывающийся архив, содержащий все компоненты устанавливаемого программного обеспечения, в который, как правило, входит большое число различных библиотек, рисунков, являющихся частью интерфейса, и так далее. Это сильно увеличивает размер таких файлов. При этом чаще всего большие установочные пакеты – легальные продукты. Хотя, встречаются случаи, когда злоумышленники вставляют зловредные файлы в состав установочных пакетов, маскируя их таким образом.

Подводя итог можно сказать, что среди исполняемых файлов часто встречаются группы схожих по функционалу файлов, отличающихся от прочих своими размерами. Комбинируя этот признак с другими можно получить неплохие эвристики.

Заметим, что размер файла – признак не бинарный. В будующем предполагается провести бинаризацию данного признака.

## 4.3 Признаки и эвристики

При извлечении признаков этого типа предполагается применение различных эвристик, полученных на основе знаний в предметной области. Эту группу признаков планируется активно пополнять новыми эвристиками для более успешной работы будущего классификатора исполняемых файлов.

В данной работе мы рассмотрим несколько признаков и подробно обсудим причины, по которым эти признаки могут быть информативными.

**Наличие связки call ... pop.** Работа этих команд достаточно подробно описана в разделе "PE формат и известные команды". В данном случае важным моментом является запись `VA` точки возврата командой `call` перед передачей управления.

Достаточно часто в исполняемых файлах встречается последовательный вызов `call` и команды `pop`, расположенной по адресу передачи управления. Как нетрудно заметить, после выполнения двух этих команд в одном из регистров окажется `VA` точки возврата из процедуры, или, иначе говоря, `VA` инструкции, следующей за `call`. Зачем это может быть нужно?

Рассмотрим это на примере простейших вирусов, дописывающих свой код в конце файла и подменяющих точку входа на себя. Тело вируса, как и любой код, состоит из последовательных команд и вызовов процедур. Обычно для вызова процедуры достаточно знать её `RVA`, поскольку на этапе выполнения имеется информация о `ImageBase` и система сама вычисляет `VA`, по которому передаётся выполнение. Теперь предположим, что вызов процедуры произошёл из тела вируса. Если бы он пользовался аддресацией относительно `ImageBase` заражённого файла, ему бы пришлось каждый раз при заражении пересчитывать все `RVA`, входщие в состав его кода. Это связано с тем, что файлы имеют разный размер, как физический, так и виртуальный.

Для избежания подобных проблем, вирусы поступают следующим образом. Все `RVA` они отсчитывают относительно некоторй *базовой* инструкции, находящейся в начале их кода. В этом случае все `RVA` процедур, используемых вирусом, статичны от одного заражённого файла к другому. Для выполнения тела вируса остаётся одна задача – определить `VA` этой самой базовой инструкции. Тут им и помогает связка `Call pop`. Вставив её в начале своего кода они получают `VA`, неободимый им для дальнейшей работы.

Однако легальные программы также могут пользоваться этой комбинаций команд. Например, многие *упаковщики* используют её для тех же целей, что и вирусы. Упаковщики – программы, основной задачей которых является шифрование или сокращение физического размера исполняемого файла. При исполнении запакованного файла вначале происходит работа тела распаковщика, которое внедряется в файл на этапе паковки. В этом смысле упакованные файлы чем-то похожы на файлы, заражённые вирусом.

В любом случае, наличие в файле связки `call ...pop` в большинстве случаев подозрительно.

**Процесс поиска** в файле связки команд `call` и `pop` происходит достаточно просто. Предлагается совершить линейный проход файла. При встрече кода операции `call` берётся аргумент этой команды. Далее проверяется, какая операция находится по адресу перехода. Если мы находим там код операции `pop`, то автоматически приравниваем признак единице.

Заметим, что рассмотренный способ в некоторых случаях может допускать ошибки. Это связано с вероятностью случайного присутствия искомой комбинации байт в файле, не имеющего отношения к командам `call` и `pop`. Однако предполагается, что эта вероятность пренебрежимо мала.

**Частота вызовов call и jmp.** Было принято решение считать **число встречающихся в файле команд** `call` и `jmp`.

Эта эвристика основана на том, что достаточно часто код обфусцируется именно с помощью этих команд. Код при этом разрывается на части, которые беспорядочно переставляются местами, и переход от одной части к другой осуществляется вставляемыми командами `jmp` или `call`. В таких файлах наблюдается достаточно высокое число вызовов этих команд, что отличает их от необфусцированных файлов.

**Расположение точки входа.** Как было сказано, вирусы часто заражают файлы, дописывая свой код в конец и меняя точку входа на себя. В этом случае точка входа будет находиться где-то внизу файла. Чистые же программы, в большинстве случаев, имеют точку входа в верхней части файла. Упомянутые выше упаковщики также достаточно часто имеют точку входа внизу файла.

Пользуясь этими соображениями было принято решение извлекать непрерывный признак, равный отношению `RVA` точки входа к виртуальному размеру файла.

**Плотность файла.** Эта эвристика – пока что единственная в моей работе попытка бороться с пакованными файлами.

Есть большое количество ситуаций, в которых знание о запакованности файла может стать решающим. В качестве примера приведём всё тот же руткит. Опыт показывает, что в большинстве случаев упакованный драйвер, импортирующий `KeServiceDescriptorTable` – руткит.

Для того чтобы понять, запакован файл или нет, существует много известных подходов. Одни основаны на вычислении энтропии файла, другие – на плотности ненулевых байт в файле или в его частях.

В данной работе для простоты подсчитывалась суммарная плотность ненулевых байт в файле. Предполагается, что данный признак в сочетании с другими извлекаемыми признаками сможет сказать что-то о запакованности файла.

**Частота команды xor.** Этот признак извлекается исходя из следующих соображений.

Если зловредный файл использует шифрование с ключом, то в нём скорее всего будет высокий уровень числа команды `xor`.

Также с помощью команды `xor` часто происходит обфускация кода. При этом, например, вместо команды `mov eax,0` используется множественный вызов команды `xor eax, eax`.

**Число секций.** Как было упомянуто выше, одним из наиболее распространённых способов заражения PE файлов является дописывание кода вируса в конец файла. Достаточно часто вирусы для этого создают дополнительную секцию, в которую и

записывают свой код. Таким образом, файлы с большим числом секций могут быть для нас подозрительны.

**Имена секций.** Этот эвристический признак основан примерно на тех же соображениях, что и прошлый.

Используя имена секций также можно установить, на каком языке программирования написан файл или каким упаковщиком он запакован.

**Число исполняемых секций.** У каждой секции исполняемого файла есть соответствующий ей набор флагов. Эти наборы находятся в структурах, описывающих секции. Среди прочих в этом наборе есть флаг, определяющий возможность исполнения кода, находящегося в данной секции.

В большинстве случаев весь код исполняемого файла находится в одной секции. Если файл заражён вирусом, то высока вероятность наличия двух исполняемых секций.

Таким образом, наличие в файле двух и более исполняемых секций – признак, указывающий на возможное заражение данного файла инфектором.

**Наличие overlay** Напомним, что overlay – часть PE файла, находящаяся за концом последней секции. При загрузке приложения в память эта часть игнорируется.

Достаточно часто зловредные файлы хранят используемую ими информацию именно в overlay. Послу загрузки в память они могут обращаться к собственному телу на жёстком диске и считывать необходимую информуцию из overlay.

Наличие overlay, с другой стороны, может свидетельствовать о том, что перед нами – самораспаковывающийся архив. Многие самораспаковывающиеся архивы хранят свои ресурсы именно в этой области файла.

Так или иначе, присутствие оверлея – интересный признак. Его комбинация с другими признаками может дать неплохие рещультаты.

Например, наличие в файле overlay и строки `NullSoft` практически однозначно указывает на то, что перед нами – самораспаковывающийся `NSIS` архив (Nullsoft Scriptable Install System).

# 5 Эксперимент

В рамках изучения поставленной задачи был программно реализован алгоритм извлечения признаков из PE файла. В нём реализованы не все признаки, описанные в прошлой главе.

В этой главе будут представлены результаты работы написанной программы.

## 5.1 Извлечение признаков

В программе было реализовано извлечение следующих признаков из исполняемого файла: тип файла, импорт файла, наличие экспорта у файла, строки файла, присутствие связки `call...pop`, частота команды `jmp`, частота команды `call`, расположение точки входа в файле, плотность файла.

Бинаризация непрерывных признаков реализована не была.

**Описание данных** Для настройки алгоритма извлечения признаков была собрана небольшая выборка из 382 чистых файлов и 643 зловредных файлов. В состав чистых файлов вошли как системные файлы ОС Windows, так и файлы различного легального программного обеспечения. В состав зловредных файлов случайным образом набирались разные виды троянов и червей. В рамках данного эксперимента заражённые вирусами файлы рассмотрены не были.

Все данные собирались в вирусной коллекции компании ЗАО "Лаборатория Касперского".

**Построение набора импортируемых функций.** Начальное множество API функций, из которых впоследствии выбирался итоговый список искомых API функций, состояло из имён всех функций, экспортируемых библиотеками `kernel32.dll`, `user32.dll`, `shell32.dll`, `wininet.dll`, `urlmon.dll`, `ws2_32.dll`, а также файлом `ntoskernel.exe`.

На основе информации о частоте встречаемости упомянутого множества API функций среди чистых и зловредных файлов выборки был сформирован список из 109 API функций, в который поровну вошли самые "популярные" функции среди чистых файлов и самые "популярные" среди зловредных файлов. В их число вошли такие функции, как `Process32First`, `Process32Next`, `TerminateProcess`, `CopyFile`, `CreateFile`, `GetProcAddress`, `LoadLibrary` и многие другие.

Отметим, что первые 3 из перечисленных функций используются зловредными файлами для завершения искомого процесса – обычно запущенного в системе антивируса.

**Построение набора искомых строк.** Этот процесс происходил абсолютно также, как и составление списка API функций. Брались всё те же выборки и из них извлекались строки. Было принято решение в качестве списка искомых строк отбирать самые часто встречаемые среди зловредных программ строки. Это решение основано на предположении, что строки, встречаемые в чистых файлах, в большинстве случаев могут встречаться и в зловредных файлах. Обратное утверждение, как правило, неверно. Таким образом был получен список из 164 "зловредных" строк. Перечислим 10 самых часто встречавшихся среди зловредных файлов выборки строк.

| | |
|---|---|
| 1. | `Software\Microsoft\Windows\CurrentVersion\Run` |
| 2. | 127.0.0.1 |
| 3. | width |
| 4. | login |
| 5. | NICK |
| 6. | advapi32.dll |
| 7. | user32.dll |
| 8. | 360tray.exe |
| 9. | svchost.exe |
| 10. | explorer.exe |

Мы получили очень интересные результаты. Скажем несколько слов о некоторых из этих строк.

Первая строка – ключ реестра, в котором записаны приложения, автоматически запускаемые при старте системы. Зловредные файлы регулярно записывают себя в

этот ключ для того, чтобы постоянно присутствовать в памяти системы.

Вторая строка – не что иное, как IP адрес *локального хоста*. Он используется в тех случаях, когда, например, клиентская и серверная часть сервиса находится на одном компьютере.

По представленному списку можно сделать вывод, что в выборке было достаточно большое число Backdoor.Win32.IRCBot. Это Backdoor, управление которого производится злоумышленником через IRC канал. Такие выводы сделаны на основе строк login и NICK, занявших высокие позиции в списке.

На восьмой позиции мы наблюдаем имя исполняемого модуля антивируса 360Tray. Это обусловлено желанием зловредных файлов перед началом своей работы "расчистить" себе путь и завершить запущенные в системе процессы антивирусов.

В завершение скажем, что имена системных процессов ОС Windows, занявшие 9 и 10 позиции, могут быть нужны зловредным файлам, как правило, для внедрения своего кода в часть памяти, занятую этими процессами. Таким образом зловредный код сможет исполняться с нужными ему правами.

**Результаты работы эвристик.** Посмотрим, каким образом повели себя полученные в ходе работы эвристические признаки на реальной выборке.

В таблице представлены результаты работы эвристических признаков на рассматриваемой выборке. Для непрерывных признаков приведены их среднее (mean), максимальное (max) и минимальное (min) значение в рамках чистых (Ч) и зловредных (З) файлов. Для бинарных признаков в отдельной графе указана доля (p) объектов чистой (Ч) и зловредной (З) частей выборки, покрываемых данным признаком.

| Эвристика | min(Ч) | max(Ч) | mean(Ч) | min(З) | max(З) | mean(З) | p(Ч) | p(З) |
|---|---|---|---|---|---|---|---|---|
| packed | 0,07681 | 0,98785 | 0,73638 | 0,0190 | 0,99864 | 0,57168 | - | - |
| jmp rate | 0 | 42372 | 779 | 0 | 24258 | 413 | - | - |
| call rate | 0 | 82919 | 3388 | 0 | 97759 | 2384 | - | - |
| call...pop | - | - | - | - | - | - | 0.007 | 0.37 |
| EP | 0.003 | 0.9927 | 0,4294 | 0.0001 | 0.9987 | 0,2639 | - | - |

Как мы видим, признак, основанный на поиске связки команд `call ...pop`, выделил достаточно большую долю зловредных файлов. Доля выделенных им чистых файлов чрезвычайно мала по сравнению с ней. Это указывает на высокую информативность данного признака, по крайней мере в рамках рассматриваемой выборки.

Вопреки ожиданиям, в в чистых файлах точка входа в среднем располагалась ниже, чем у зловредных файлов. Это может быть связано с тем, что в выборке не было представлено заражённых вирусами файлов.

Плотность чистых файлов превосходит плотность зловредных предположительно по той причине, что среди чистых файлов встречалось много установочных пакетов, хранящих свои компоненты в запакованном виде.

# 6 Заключение

В данной работе представлен новый подход к задаче извлечения признаков из исполняемых файлов. Его основная идея состоит в извлечении бинарных эвристических признаков, получаемых с учётом глубоких знаний об устройстве и свойствах зловред-

ных файлов. Это отличает его от всех предыдущих работ, посвящённых вопросам статического анализа и классификации исполняемых файлов. В них авторы извлекали в большинстве случаев малоинформативные признаки, лежащие на поверхности PE формата. Это приводило к необходимости генерации избыточного числа признаков, что в свою очередь замедляло работу их алгоритмов.

В ходе решения задачи было предложено некоторое количество эвристических признаков, на основе которых в будущем планируется решать задачу классификации исполняемых файлов. Некоторые из них в ходе поставленного эксперимента проявили неплохую способность выделять зловредные файлы.

Также был программно реализован алгоритм извлечения признаков из PE файлов. Алгоритм устроен таким образом, что есть возможность легко добавлять в него новые эвристики.

В будущем планируется пополнять множество эвристиеских признаков и, набрав необходимое число признаков, приступить непосредственно к задаче классификации исполняемых файлов.

# Список литературы


[1] M. Christodorescu, S. Jha. Static Analysis of Executables to Detect Malicious Patterns. — In Proceedings of the 12th USENIX Security Symposium, 2003.

[2] J. Z. Kolter, M. A. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. — Journal of Machine Learning Research 7, p.:2721-2744, 2006.

[3] M. G. Schultz, E. Eskin, E. Zadok, S. J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. —In Proceedings of the IEEE Symposium on Security and Privacy, p.: 38 – 49, Oakland, CA, 2001.

[4] M. A. Siddiqui. Data Mining Methods for Malware Detection. — A dissertation for the degree of Doctor of Philosophy, Florida, 2008.

[5] Kaspersky Lab. Homepage — kasprsky.com

[6] Information About Viruses, Hackers and Spam. Homepage — viruslist.com.


.