В. Ф. Макаров, П. В. Румянцев

ОСНОВНЫЕ МЕТОДЫ ИССЛЕДОВАНИЯ ПРОГРАММНЫХ СРЕДСТВ СКРЫТОГО ИНФОРМАЦИОННОГО ВОЗДЕЙСТВИЯ

Введение

Современный мир является крайне сложной системой, в которой заметную роль играют компьютеры, связанные между собой посредством каналов связи в локальные и глобальную сети. Результатом повсеместного внедрения компьютеров в повседневную жизнь и деятельность человека является огромный круг проблем, связанный с безопасностью информации, циркулирующей в компьютерных системах, а также с надежностью функционирования таких систем. Характерной частью компьютерной системы является то, что стоимость информации, циркулирующей в ней, намного выше стоимости оборудования. Естественно, что на компьютерную систему осуществляются атаки с целью овладеть информацией, исказить ее или же сделать недоступной для владельца. В данной статье приводится обзор методов скрытого воздействия на информацию, циркулирующую в компьютерной системе, а также методов обнаружения скрытых воздействий.

1. Понятие программного средства скрытого информационного воздействия (ПССИВ)

Под программными средствами скрытого информационного воздействия, которые могут находиться в исполняемом файле, можно понимать:

- ошибки автора программы, сделанные случайно и позволяющие нарушить нормальное функционирование компьютерной системы;
- участки кода программы, написанные автором программы с целью нарушения нормального функционирования компьютерной системы;
- участки «навесного» кода программы, написанные после выпуска программы злоумышленником, не являющимся автором программы, с целью нарушения нормального функционирования компьютерной системы, например вирусы;
- «навесное» программное обеспечении, маскирующее наличие в программе участков, обеспечивающих нормальное функционирование компьютерной системы, например упаковщики, и т. д.

Все скрытые информационные воздействия оставляют в исполняемом файле некоторые следы, по которым они могут быть обнаружены.

Под программной закладкой можно понимать последовательность команд, которая была включена в программу на этапе ее разработки, реализующую некий алгоритм, обеспечивающий получение злоумышленником конфиденциальной информации или ограничивающий функциональность системы, в которой эти команды исполняются.

У программных закладок нет общих признаков, позволяющих однозначно идентифицировать их именно как закладки. Их можно идентифицировать только по косвенным признакам, содержащимся непосредственно в их коде, например по обращению к файлам, в которых содержится важная информация, по попыткам открыть сетевые соединения и т. д.

Разработчик программной закладки может сознательно замаскировать ее при помощи специальных способов. К таким способам могут быть отнесены отказ от использования стандартных механизмов работы и замена их собственными (например, разработка канальных программ для доступа к внешним устройствам) и использование различных технологий полиморфизма, т. е. применение различных кодовых конструкций для достижения одной и той же цели.

2. Демаскирующие ПССИВ признаки

Демаскирующие ПССИВ признаки можно разделить на несколько групп.

- признаки, содержащиеся в метаданных файла, т. е. в его заголовках и данных о размещении различных его частей;

- признаки, находящиеся в собственном (не экспортируемом) коде файла;
- аномалии, т. е. какие-то характеристики, отличающиеся от подобных характеристик в большинстве файлов.

2.1. Признаки, демаскирующие применение маскировочных ПССИВ

Признаки, демаскирующие применение маскировочных ПССИВ, в наибольшей степени могут быть подвергнуты формализации. Как следует из сказанного ранее, основной задачей маскировочных ПССИВ является изменение в максимальной степени «внешнего вида» исполняемого файла с целью противодействия его статическому анализу. Решение этой задачи практически невозможно без изменения метаданных файла.

Как показывает практика, формат PE исполняемых файлов не только не защищен от несанкционированного изменения, но, наоборот, словно специально сконструирован таким образом, чтобы дать злоумышленнику широкие возможности по модификации файла.

С одной стороны, внутри файла формата PE с весьма высокой вероятностью находятся каверны, в которые можно записать исполняемый код. С другой стороны, без особых проблем исполняемый код можно изменить до неузнаваемости, сделав его самомодифицирующимся. И, к сожалению, механизм, предоставляющий возможность подписать PE-файл (signtool.exe), на практике используется очень редко. Более того, проверку подписи можно легко отключить при помощи корректуры «необязательного» заголовка PE-файла.

Кроме того, системный загрузчик не проверяет правильность заполнения большинства полей заголовков файлов, поэтому в непроверяемых полях можно разместить какой-либо исполняемый код или же какие-либо данные, используемые программой-маскировщиком.

В качестве примера приведем специальным образом подготовленный файл. Его шестнадцатиричное представление приведено на рис. 1.

```
00000000  4D 5A 2E 2E 50 45 00 00 4C 01 01 00 4D 65 73 73  MZ..PE..L...Mess
00000010  61 67 65 42 6F 78 41 00 40 00 0F 01 0B 01 57 B8  ageBoxA.@.....Wё
00000020  44 00 80 00 50 2C 38 50 57 EB 05 90 1E 00 00 00  D.Ђ.P,8PWл.ђ....
00000030  FF 15 8C 00 80 00 C3 00 00 00 80 00 04 00 00 00  я.Ь.Ђ.Г...Ђ.....
00000040  04 00 00 00 75 73 65 72 33 32 00 00 04 00 00 00  ....user32......
00000050  00 00 00 00 99 00 00 00 88 00 00 00 00 00 00 00  ....™...€.......
00000060  02 00 00 00 11 00 00 00 88 00 00 00 11 00 00 00  ........€.......
00000070  88 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00  €...............
00000080  20 00 30 60 88 00 00 00 00 00 00 00 0A 00 00 00  .0`€............
00000090  00 00 00 00 44 00 00 00 8C                       ....D...Њ
```

Рис. 1. Шестнадцатиричный дамп специальным образом подготовленного файла

Видно, что значение поля e_lfanew MZ-заголовка (смещение 0x3c) равно 0x04, т. е. меньше 64 (0x40). В этом случае можно сделать однозначный вывод о том, что на файл произведено воздействие.

В файле, приведенном в качестве примера (рис. 1), размер поля SizeOfOptionalHeader, находящегося в данном случае по смещению 24 (0x18), составляет 64 (0x40), а не 224 (0xe0), как в обычных файлах. Этот факт позволяет сделать вывод о том, что на файл произведено воздействие.

Интересно, что в файле, приведенном в качестве примера (рис. 1), значение поля AddressOfEntryPoint равно 30 (0x1e), что существенно меньше размера MZ-заголовка. Это позволяет сделать однозначный вывод о том, что на файл произведено воздействие.

Например, компилятор Microsoft определяет следующие стандартные имена секций: «.arch», «.bss», «.data», «.edata», «.idata», «.pdata», «.rdata», «.reloc», «.rsrc», «.sbss», «.sdata», «.srdata», «.text», «.xdata». В том случае, если в файле встречается имя секции, отличное от стандартного, можно говорить о том, что на исполняемый файл произведено какое-либо воздействие.

На рис. 2 приведен пример файла chess.exe, входящего в стандартную поставку Windows Vista. Исходя из того, что в файле присутствует секция с именем, отличным от стандартного, можно сделать вывод о том, что на файл было произведено воздействие.

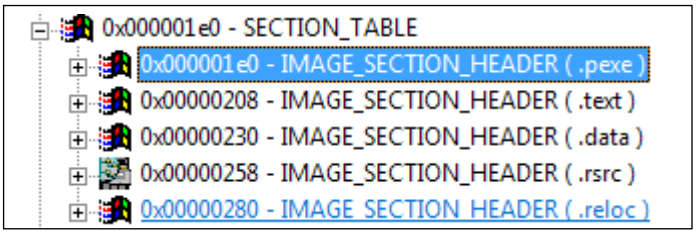

Рис. 2. Файл с нестандартной секцией

Кроме того, некоторые программы (упаковщики, протекторы и крипторы), которые могут быть использованы как маскировщики, при изменении файла создают секции с хорошо известными именами. Так, например, упаковщик ASPack создает в файле секции с именами «.aspack» и «.adata». Таким образом, зачастую имя секции в некоторой степени является и идентификатором программы, произведшей воздействие на исполняемый файл.

Поле Characteristics заголовка секции содержит характеристики данных, находящихся в секции. Зачастую совокупность характеристик может свидетельствовать о том, что на файл произведено воздействие. Например, допустим, что секция, с одной стороны, содержит исполняемый код (установлены флаги IMAGE_SCN_CNT_CODE и IMAGE_SCN_MEM_EXECUTE), а с другой стороны, в секцию разрешена запись (установлен флаг IMAGE_SCN_MEM_WRITE). В этом случае естественно предположить, что в процессе выполнения код будет изменяться, другими словами, секция содержит самомодифицирующийся код.

«Необязательный» заголовок файла заканчивается массивом DataDirectory, состоящим, как правило, из 16 (0x10) элементов. При этом каждый элемент этого массива соответствует структуре IMAGE_DATA_DIRECTORY и определяет размещение в памяти данных конкретного назначения. Например, элемент с индексом 0 определяет размещение в памяти таблицы экспорта, элемент с индексом 1 — таблицы импорта, элемент с индексом 2 — таблицы ресурсов и т. д.

В связи с тем, что каждая из директорий имеет свой формат, целесообразно остановиться на тех директориях, содержимое которых может повлиять на работу системы.

Директория, соответствующая индексу 0 (IMAGE_DIRECTORY_ENTRY_EXPORT), содержит в себе таблицу экспорта, т. е. данные, позволяющие определить порядковый номер и, при наличии, имя экспортируемой функции, а также адрес этой функции.

Для автоматизированного анализа наиболее важным признаком является несоответствие значений полей NumberOfFunctions и NumberOfNames, свидетельствующее о нежелании автора раскрывать имена своих функций, что, соответственно, значительно затрудняет анализ файла.

Анализ наименований функций, к сожалению, в настоящее время формализации не поддается. Однако при беглом просмотре некоторые имена функций могут обратить на себя внимание исследователя. Например, имя функции FormatDisk сразу же должно заставить исследователя насторожиться.

При проведении системного анализа важно также определить, все ли экспортируемые данным модулем функции импортируются другими модулями. Если в таблице экспорта присутствуют функции, которые не импортируются ни одним из модулей системы, то можно сделать вывод либо о неаккуратности программиста, либо о том, что неимпортируемая функция может быть импортирована после обновления системы, например, после автоматической установки обновлений.

Директория, соответствующая индексу 1 (IMAGE_DIRECTORY_ENTRY_IMPORT), содержит в себе таблицу импорта, т. е. данные, позволяющие определить перечень импортируемых библиотек, а также перечень импортируемых из каждой библиотеки функций.

При анализе таблицы импорта необходимо определить, во-первых, какие библиотеки импортирует данный модуль. В том случае, если модуль установлен в защищенной системе, то все импортируемые библиотеки должны быть сертифицированы. Если же модуль производит импорт несертифицированной библиотеки, это сразу же должно натолкнуть исследователя на мысль о том, что в защите системы возможна брешь.

Во-вторых, необходимо внимательно присмотреться и к списку импортируемых функций. Если, скажем, безобидное на первый взгляд приложение использует функцию FormatDisk, то это также должно привлечь внимание исследователя.

Кроме того, если функция импортируется по порядковому номеру, а не по имени, это должно означать, что у автора экспортируемой функции были основания не публиковать имя функции и, соответственно, затруднить анализ программы и процесс понимания функциональности программы исследователем. В этом случае исследователь не должен ограничиваться структурным анализом, а произвести также функциональный анализ функции, экспортируемой библиотекой.

К сожалению, при помощи таблицы импорта производится так называемый неявный (implicit) импорт. Однако программист может воспользоваться также и явным (explicit) импортом при помощи функции LoadLibrary, затем определить адрес интересующей функции при помощи функции GetProcAddress, после чего напрямую по указателю обратиться к данной функции. Задача выявления характеристик явного импорта может быть решена при помощи функционального анализа.

Зачастую программисты, желающие скрыть функциональность программы от исследователей, применяют различные способы модификации программ. При этом, как правило, модифицированная программа импортирует только библиотеку kernel32.dll, из которой импортируются только функции LoadLibrary и GetProcAddress. Иногда могут импортироваться и некоторые другие функции, такие, например, как VirtualAlloc и VirtualFree. Если дело обстоит именно так, то исследователю необходимо исходить из того, что исследуемый модуль защищен.

Директория, соответствующая индексу 2 (IMAGE_DIRECTORY_ENTRY_RESOURCE), содержит в себе ресурсы программы, т. е. некоторые данные, которые могут использоваться программным кодом. Говоря более точно, эта директория содержит в себе некоторые метаданные, описывающие расположение ресурсов в исполняемом файле, а также сами ресурсы.

Но у ресурсов есть одна особенность, связанная с тем, что программист может самостоятельно определять типы данных и их формат. Например, в секции типа RC_DATA может находиться другой исполняемый файл, который будет включать в себя ПССИВ.

2.2. Признаки, демаскирующие применение непосредственных ПССИВ, и способы их выявления

В отличие от признаков, демаскирующих применение маскировочных ПССИВ, формализовать признаки применения непосредственных ПССИВ достаточно сложно, так как невозможно создать полную и всеобъемлющую базу признаков наличия ПССИВ в программном модуле. В некотором смысле ПССИВ можно сравнить с вирусами. В настоящее время для поиска ПССИВ, как и для поиска вирусов, используют три технологии — сигнатурный анализ, эвристический анализ и поведенческий анализ.

Сигнатурный анализ является наиболее простым средством. Допустим, что ранее во время анализа файлов, содержащих ПССИВ, были обнаружены некоторые последовательности байтов, находящиеся в коде, при помощи которых можно однозначно идентифицировать находящееся в файле ПССИВ. В этом случае можно сказать, что наличие в файле данной последовательности (сигнатуры) однозначно свидетельствует о нахождении в файле ПССИВ.

Однако этот способ анализа имеет и несколько ограничений. Во-первых, он может быть эффективным только в тех случаях, когда ПССИВ является широко распространенным (например,

вирус), ранее проанализированным и, соответственно, когда сигнатура ПССИВ находится в базе данных сигнатур. Во-вторых, автор ПССИВ, внеся небольшие «косметические» изменения в ПССИВ, например сделав ПССИВ полиморфным, может легко «обмануть» анализатор.

Эвристический анализ является более сложным, но и, возможно, более эффективным видом анализа. В данном случае анализируются не сигнатуры, а некоторые последовательности команд, характерных для ПССИВ. Например, такими последовательностями могут быть команды записи в определенные ветви реестра, открытие и попытки записи в важные файлы и т. д. Одна команда сама по себе не может служить признаком ПССИВ, однако последовательности команд позволяют сделать вывод о присутствии ПССИВ в коде анализируемого модуля. По имеющимся данным, первая попытка создания модуля эвристического анализа была предпринята в 1993–1994 г. программистом Евгением Сусликовым, не являвшимся в то время вирусологом, в программе Lie Detector (Детектор Лжи). Именно с того времени модули эвристического анализа являются неотъемлемой частью всех более или менее серьезных антивирусных программ.

Эвристический анализ, в отличие от сигнатурного анализа, позволяет обнаружить не только известные ранее, но и неизвестные ПССИВ, однако его недостатком является большое число ложных срабатываний.

И, наконец, поведенческий анализ. Его часто также называют эвристическим анализом, однако, по мнению авторов этой статьи, эвристический анализ является видом статического анализа, а поведенческий анализ более близок к анализу динамическому, позволяющему при помощи эмуляции кода распознать подозрительное поведение программного модуля, содержащего ПССИВ.

2.3. «Аномалии» в файлах

Под аномалиями в данном случае можно понимать некоторые характеристики, присутствующие в исследуемом файле и не присутствующие в большинстве файлов того же типа. На рис. 3 приведено побайтовое представление обычного исполняемого файла (байты размещены по спирали, цвет пикселя соответствует значению байта), хорошо видны составные части файла в центре рисунка.

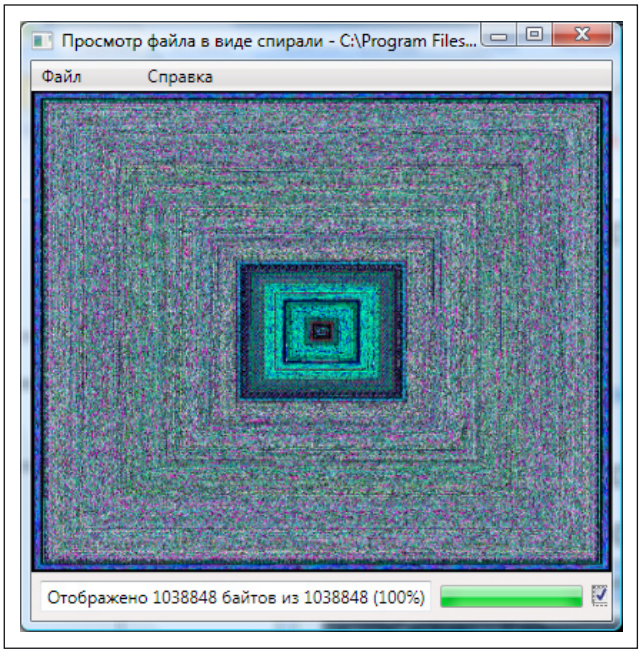

Рис. 3. Побайтовое представление «нормального» файла

На следующем рисунке (Рис. 4) приведено такое же представление исполняемого файла, подвергшегося воздействию.

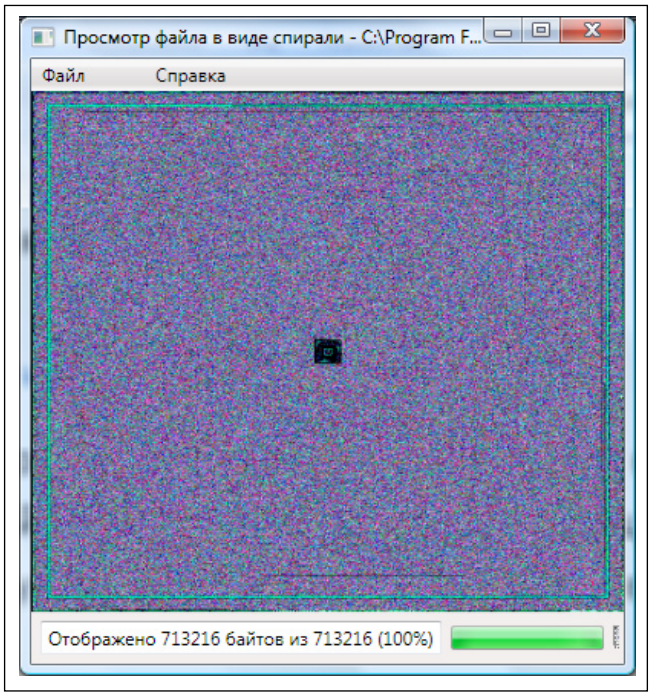

Рис. 4. Файл, подвергшийся воздействию

Видно, что на первом рисунке информация имеет некоторую структуру (характерную для исполняемых файлов). В то же время информация, представленная на втором рисунке, никакой структуры не имеет. Именно это в данном случае и является аномалией.

В общем случае наличие аномалий в файле формализовано быть не может. Аномалии проявляются при различных проекциях файлов в ходе «ручного» анализа файла при помощи специально разработанных для этого средств.

3. Требования к программам обнаружения ПССИВ в автоматическом режиме

Исходя из сказанного выше можно сформулировать некоторые требования, предъявляемые к программам обнаружения ПССИВ. Эти требования отличаются от требований, предъявляемых, скажем, к антивирусным программам.

Во-первых, пользователем антивирусной программы предполагается человек, не имеющий специальных знаний и не имеющий возможности принять решение о наличии в файле вируса даже в том случае, когда у него есть вся необходимая для этого информация. Пользователем средства обнаружения ПССИВ предполагается человек, обладающий обширными знаниями, позволяющими на основе полученных данных сделать вывод о том, что в файле присутствует ПССИВ.

Во-вторых, для антивируса желательно уменьшить число ложных срабатываний. Для программы поиска ПССИВ, наоборот, чем больше признаков наличия ПССИВ обнаружено, тем лучше, тем большую информацию об исследуемом файле и об исследуемой системе получает исследователь. Проведя анализ файлов в системе, пользователь этого средства должен получать некую базу кумулятивных весовых коэффициентов файлов, построенную на основе базы знаний о признаках наличия ПССИВ в файле. При этом больший весовой коэффициент соответствует большей степени угрозы, исходящей от файла.

Кроме того, после завершения работы антивируса работа обычного пользователя завершается, при этом пользователь должен оставаться в уверенности, что никаких вирусов в системе нет. А после завершения работы средства обнаружения ПССИВ работа исследователя-пользователя этого средства только начинается. Получив список кумулятивных весовых коэффициентов

исследуемых файлов, пользователь определяет порядок, в соответствии с которым он будет исследовать файлы при помощи интерактивных программ анализа файлов.

СПИСОК ЛИТЕРАТУРЫ:

1. *Румянцев П. В.* Исследование программ Win32: до дизассемблера и отладчика. М.: Горячая линия — Телеком, 2004.
2. *Щербаков А. Ю.* Разрушающие программные воздействия. М.: Эдель, 1993.
3. *Расторгуев С. П., Дмитриевский Н. Н.* Искусство защиты и «раздевания» программ. М.: Софтмаркет, 1991.
4. *Казарин О. В.* Безопасность программного обеспечения компьютерных систем. Монография. М.: МГУЛ, 2003. — 212 с.
5. *Касперский Е.* Компьютерное Zловредство. СПб.: Питер, 2008.