1 Основные статические методики

Мы начнем знакомиться с анализом вредоносного ПО со статических методик, которые, как правило, применяются в первую очередь. *Статический анализ* заключается в исследовании кода или структуры программы для определения ее функций. Сама программа в это время не запущена. Для сравнения, при выполнении *динамического анализа* аналитик обычно запускает программу (подробнее об этом — в главе 3 «Основы динамического анализа»).

В этой главе обсуждаются разные способы извлечения полезной информации из исполняемых файлов. Будут рассмотрены следующие приемы.

- Подтверждение вредоносности с помощью антивируса.
- Использование хешей для идентификации вредоносов.
- Сбор информации из строк, функций и заголовков файла.

Все эти методики позволяют получить разные сведения, и то, какую из них следует выбрать, зависит от ваших целей. Обычно используется сразу несколько методик, чтобы собрать как можно больше информации.

Сканирование антивирусом: первый шаг

Анализ предполагаемой вредоносной программы можно начать с использования нескольких антивирусов, которые, возможно, уже знакомы с ней. Однако антивирусные пакеты далеки от идеала. Они в основном полагаются на базу данных известных участков кода *(файловых сигнатур)*, а также выполняют поведенческий анализ и сопоставление по образцу *(эвристический подход)*, чтобы распознавать подозрительные файлы. Проблема в том, что авторы вредоносного ПО могут легко модифицировать свой код, изменяя тем самым сигнатуры своих программ и оставаясь незаметными для антивирусных сканеров. Кроме того, редкие вредоносы часто избегают обнаружения, так как они попросту еще не внесены в базу данных. И наконец, эвристические методики часто обнаруживают неизвестные вредоносные программы, но могут пропустить новый и уникальный код.

Разные антивирусы используют разные сигнатуры и эвристики, поэтому бывает полезно применить к одному и тому ж файлу сразу несколько из них. Такие сайты, как VirusTotal (www.virustotal.com), позволяют просканировать файл разными антивирусными системами. VirusTotal генерирует отчет, в котором отмечает, сколько антивирусов считает файл вредоносным, и указывает название вредоноса и дополнительную информацию о нем, если таковая имеется.

Хеширование: отпечатки пальцев злоумышленника

Хеширование — это распространенный метод однозначной идентификации вредоносного ПО. Зараженный файл пропускается через программу хеширования, в результате чего получается уникальная строка (хеш), которая служит идентификатором вредоноса. Одной из наиболее распространенных функций хеширования, применяемых аналитиками безопасности, является MD5, хотя SHA-1 тоже пользуется популярностью.

Например, если запустить программу md5deep (которая находится в свободном доступе), чтобы вычислить хеш приложения Пасьянс, поставляемого вместе с Windows, получится следующий результат:

```
C:\>md5deep c:\WINDOWS\system32\sol.exe
373e7a863a1a345c60edb9e20ec3231 c:\WINDOWS\system32\sol.exe
```

Хеш равен `373e7a863a1a345c60edb9e20ec3231`.

На рис. 1.1 показано графическое приложение WinMD5, которое умеет вычислять и отображать хеши сразу для нескольких файлов.

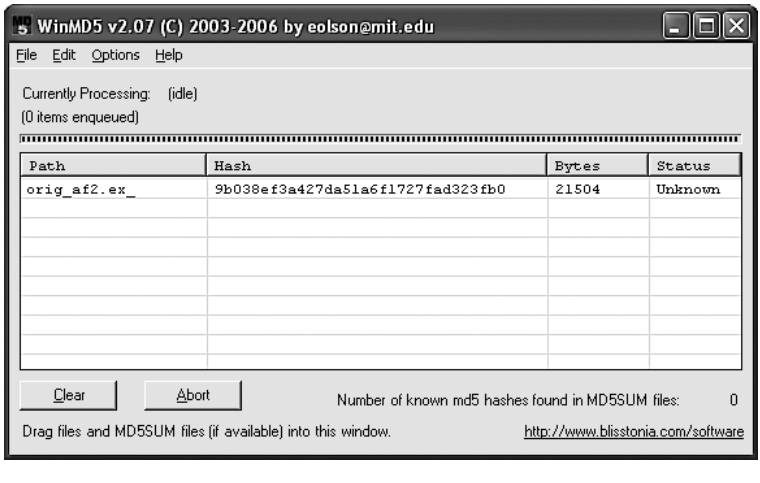

Рис. 1.1. Вывод программы WinMD5

Уникальный хеш, принадлежащий какому-то участку вредоноса, можно использовать следующим образом:

- в качестве маркера;
- поделиться им с другими аналитиками, чтобы помочь им распознать угрозу;
- поискать его в Интернете на случай, если он уже был идентифицирован ранее.

Поиск строк

Строка в программе — это последовательность символов, например the. Если программа выводит сообщения, проходит по URL-адресу или копирует файл в определенное место, это означает, что она содержит строки.

Поиск по строкам может дать некоторое представление о функциях программы. Например, если она обращается к URL, вы найдете соответствующий адрес, хранящийся в виде строки. Строки в исполняемом файле обычно хранятся в формате ASCII или Unicode, и для их поиска можно воспользоваться программой Strings (bit.ly/ic4plL).

ПРИМЕЧАНИЕ

Компания Microsoft имеет собственную реализацию Unicode, строки в которой состоят из так называемых широких символов. В данной книге под Unicode подразумевается именно эта реализация.

И в ASCII, и в Unicode символы хранятся в виде последовательностей со значением NULL в конце *(нулевой символ)*, которое указывает на завершение строки. В ASCII каждый символ занимает 1 байт, а в Unicode — 2 байта.

На рис. 1.2 показана строка `BAD`, сохраненная в формате ASCII. Она состоит из байтов 0x42, 0x41, 0x44 и 0x00, где 0x42 обозначает `B`, 0x41 — `A` и т. д. Байт 0x00 в конце сигнализирует о завершении строки.

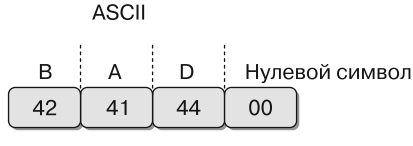

Рис. 1.2. Строка BAD, представленная в формате ASCII

На рис. 1.3 показана строка `BAD`, сохраненная в формате Unicode. Она состоит из байтов 0x42, 0x00, 0x41 и т. д. Прописная `B` представлена байтами 0x42 и 0x00, а нулевой символ выглядит как два байта 0x00 подряд.

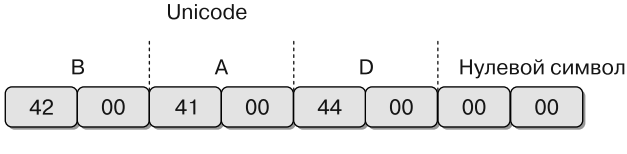

Рис. 1.3. Строка BAD, представленная в формате Unicode

Когда программа Strings ищет строки в форматах ASCII и Unicode, она игнорирует контекст и форматирование. Это позволяет ей анализировать файлы любого типа и находить строки на любых их участках (с другой стороны, она может обнаружить байты, которые не являются строкой). Искомые строки должны состоять как минимум из трех букв в формате ASCII или Unicode и завершаться нулевым символом.

Иногда строки, обнаруженные программой Strings, таковыми не являются. Например, последовательность байтов 0x56, 0x50, 0x33 и 0x00 будет интерпретирована как строка `VP3`, хотя это может быть адрес в памяти, инструкция процессора или данные, используемые приложением. Определение таких случаев ложится на пользователя.

Недействительные строки обычно легко выявить, так как они не представляют собой корректный текст. Например, в следующем отрывке показан результат выполнения программы Strings для файла `bp6.ex_`:

```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 ❶
e-@
GetLayout ❷
GDI32.DLL ❸
SetLayout ❹
M}C
Mail system DLL is invalid.!Send Mail failed to send message. ❺
```

В этом примере строки, выделенные жирным шрифтом, можно игнорировать. Обычно, если строка короткая и не соответствует никакому слову, она бессмысленна.

С другой стороны, строки `GetLayout` ❶ и `SetLayout` ❷ являются функциями, которые используются в графической библиотеке Windows. Их легко определить, так как функции Windows и последующие слова, содержащиеся внутри, обычно начинаются с большой буквы.

`GDI32.DLL` ❸ имеет значение, поскольку это имя популярной в Windows *библиотеки динамической компоновки* (dynamic link library, DLL), которая используется графическими программами (DLL-файлы содержат исполняемый код, который разделяется между разными приложениями).

Номер `99.124.22.1` ❹ является IP-адресом — одним из тех, которые будут каким-то образом использованы вредоносной программой.

Строка `Mail system DLL is invalid.!Send Mail failed to send message.` ❺ представляет собой сообщение об ошибке. Это конкретное сообщение говорит нам о двух вещах: предполагаемый вредонос передает текст (вероятно, по электронной почте) и зависит от системной динамической библиотеки для отправки писем. Исходя из этого имеет смысл проверить журнальные записи электронной почты на подозрительный трафик; кроме того, другая библиотека (`Mail system DLL`) также может быть связана с данным вредоносным кодом. Стоит отметить, что недостающий DLL-файл может оказаться безвредным; зловреды часто используют в своих целях обычные библиотеки.

Упакованное и обфусцированное вредоносное ПО

Злоумышленники часто упаковывают и обфусцируют вредоносные файлы, чтобы их было сложнее обнаружить или проанализировать. *Обфусцированными* являются программы, авторы которых пытаются скрыть их выполнение. *Упакованные* программы являются подмножеством обфусцированных; их содержимое сжато, и анализировать его невозможно. Эти два приема сильно ограничивают применение статического анализа.

Безвредные программы обычно содержат множество строк, а вот в упакованных и обфусцированных вредоносах их очень мало. Если при исследовании программы с помощью утилиты Strings обнаружится лишь несколько строк, это может значить, что она является обфусцированной или упакованной и, возможно, содержит вредоносный код. Для более точных выводов вам придется применить другие методики.

ПРИМЕЧАНИЕ

Упакованный и обфусцированный код часто содержит как минимум функции LoadLibrary и GetProcAddress, которые используются для загрузки и получения доступа к дополнительным функциям.

Упаковка файлов

Одновременно с упакованной программой запускается небольшая утилита-обертка, которая освобождает запакованный файл и запускает его (рис. 1.4). При статическом анализе упакованной программы мы можем вычленить только эту обертку (упаковка и распаковка более подробно рассматриваются в главе 18).

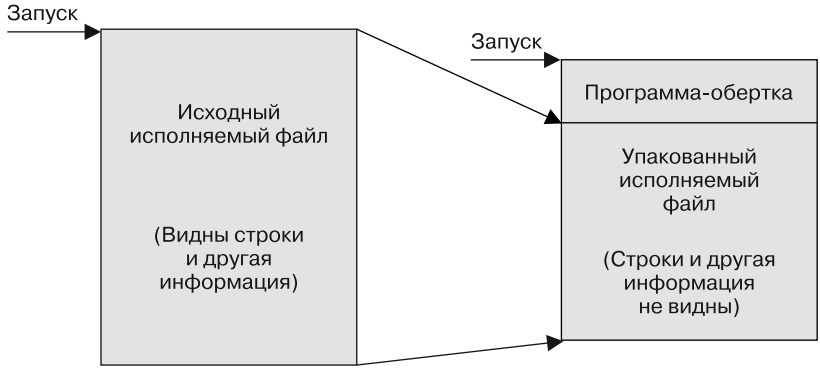

Рис. 1.4. Слева представлен исходный исполняемый файл, в котором видны все строки, инструкции импорта и другая информация. Справа показана его упакованная версия: в ней все строки, инструкции импорта и другая информация сжаты и недоступны для большинства инструментов, выполняющих статический анализ

Обнаружение упаковщиков с помощью PEiD

Определить, является ли файл упакованным, можно с помощью программы PEiD. Эта утилита позволяет установить тип упаковщика или компилятора, которые использовались при сборке приложения, что значительно упрощает анализ упакованных данных. На рис. 1.5 показана информация о файле `orig_af2.ex_`, собранная программой PEiD.

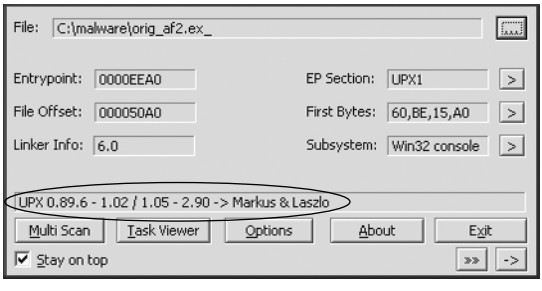

Рис. 1.5. Программа PEiD

ПРИМЕЧАНИЕ

Разработка и поддержка проекта PEiD были приостановлены в апреле 2011 года, но он до сих пор является лучшим инструментом для обнаружения упаковщиков и компиляторов. Во многих случаях он способен также определить, какой именно упаковщик использовался для заданного файла.

Утилита PEiD определила, что файл запакован с помощью UPX версии 0.89.6-1.02 или 1.05-2.90 (пока не обращайте внимания на остальную информацию: мы рассмотрим эту программу более подробно в главе 18).

Для выполнения анализа программу следует распаковать. Процесс распаковки часто оказывается сложным (описывается в главе 18), но упаковщик UPX настолько популярен и прост в использовании, что заслуживает отдельного упоминания. Например, чтобы распаковать вредоносный код, запакованный с его помощью, достаточно загрузить исполняемый файл UPX (upx.sourceforge.net) и запустить его, указав имя запакованной программы в качестве аргумента:

```
upx -d PackedProgram.exe
```

ПРИМЕЧАНИЕ

Учтите, что многие плагины к программе PEiD запускают исполняемые вредоносы без предупреждения! В главе 2 показано, как подготовить безопасную среду для выполнения вредоносного кода. Кроме того, как и любая другая программа, PEiD может содержать уязвимости. Например, версия 0.92 была подвержена переполнению буфера, что позволяло выполнять произвольный код. Умелому злоумышленнику это давало воможность написать программу, которая взломала бы компьютер аналитика безопасности. Так что всегда используйте самую последнюю версию PEiD.

Формат переносимых исполняемых файлов

До сих пор мы обсуждали инструменты, которые сканируют исполняемые файлы без учета их формата. Однако формат файла может многое сказать о возможностях программы.

Переносимый исполняемый формат (portable executable, PE) используется в Windows для исполняемых файлов, объектного кода и библиотек динамической компоновки. Формат PE — это структура данных, которая содержит информацию, необходимую системному загрузчику Windows для управления завернутым исполняемым кодом. Практически любой файл для Windows, в котором есть исполняемый код, имеет формат PE, но иногда устаревшие форматы файлов все же попадаются во вредоносных программах.

PE-файлы начинаются с заголовка, который содержит информацию о коде, типе приложения, необходимых библиотечных функциях и требованиях к дисковому пространству. Содержимое PE-заголовка представляет большую ценность для аналитика безопасности.

Компонуемые библиотеки и функции

Одним из наиболее полезных фрагментов информации, которые можно извлечь из исполняемого файла, является список функций, которые он импортирует. *Импортированные функции* — это функции, используемые одной программой, но содержащиеся в другой, например в библиотеках, которые часто хранят общие для многих программ функции. Библиотеки можно подключать к основному исполняемому файлу с помощью *компоновки*.

Программисты компонуют импортированные функции в своем коде, чтобы не повторять то, что уже было реализовано в других программах. Библиотеки можно компоновать статически, во время выполнения или динамически. Сведения о способе компоновки кода являются ключевыми для понимания работы вредоноса, поскольку от них зависит то, какую информацию можно найти в заголовке PE-файла. В этом разделе мы рассмотрим несколько инструментов для просмотра функций, импортированных программой.

Компоновка: статическая, динамическая или во время выполнения

Статический метод реже всего используется при компоновке библиотек, хотя он распространен в программах для UNIX и Linux. Если библиотека статически скомпонована с исполняемым файлом, весь ее код копируется в этот файл, что увеличивает размер итоговой программы. В ходе анализа сложно определить, какой код был статически скомпонован, а какой принадлежит самому исполняемому файлу, поскольку факт компоновки никак не упоминается в заголовке формата PE.

Компоновка на этапе выполнения не пользуется особой популярностью в обычных приложениях, но часто применяется во вредоносных программах, особенно если они упакованы или обфусцированы. Исполняемые файлы, использующие этот вид компоновки, обращаются к библиотеке только тогда, когда она им нужна, а не при запуске, как в случае с динамически скомпонованными программами.

Microsoft Windows предоставляет несколько функций, с помощью которых программисты могут импортировать код, не указанный в заголовке программы. Наиболее популярными из них являются `LoadLibrary` и `GetProcAddress`, также используются `LdrGetProcAddress` и `LdrLoadDll`. Функции `LoadLibrary` и `GetProcAddress` позволяют программе получить доступ к любой функции в любой библиотеке системы; это означает, что в случае их применения статический анализ не сможет определить, с какими функциями скомпонована подозрительная программа.

Динамический метод компоновки является самым распространенным и интересным для анализа вредоносного ПО. Если библиотеки скомпонованы динамически, операционная система ищет их при загрузке программы. Внешняя функция, которую вызывает программа, выполняется в рамках библиотеки.

Заголовок PE-файла хранит информацию обо всех библиотеках, которые будут загружены, и всех функциях, которые используются программой, — во многих случаях они являются ее самой важной частью и их идентификация имеет большое значение, позволяя предугадать поведение программы. Например, если программа импортирует функцию `URLDownloadToFile`, можно предположить, что она выходит в Интернет и загружает данные, которые затем сохраняются в локальный файл.

Исследование динамически скомпонованных функций с помощью Dependency Walker

Утилита Dependency Walker (www.dependencywalker.com), поставляемая вместе с некоторыми версиями Microsoft Visual Studio и другими пакетами разработки от Microsoft, выводит список только тех функций, которые были скомпонованы динамически.

На рис. 1.6 показаны результаты анализа файла `SERVICES.EX_` ❶ с помощью этого инструмента. На левой панели ❷ представлена как сама программа, так и DLL, которые она импортирует, а именно `KERNEL32.DLL` и `WS2_32.DLL`.

Если щелкнуть на `KERNEL32.DLL`, на правой верхней панели ❸ будет выведен список функций. Наибольший интерес представляет функция `CreateProcessA`, которая говорит о том, что программа создает другой процесс, — значит, после ее запуска нужно проследить за тем, как она запустит дополнительные программы.

На средней панели ❹ перечислены все функции библиотеки `KERNEL32.DLL`, которые можно импортировать. Для нас эта информация не особо полезна. Обратите внимание на столбцы на панелях ❸ и ❹ под названием Ordinal (Порядковый номер). Исполняемые файлы могут импортировать функции по их порядковым номерам вместо имен. В этом случае имя функции не упоминается в программе, что усложняет ее обнаружение. Когда вредоносный код применяет эту методику,

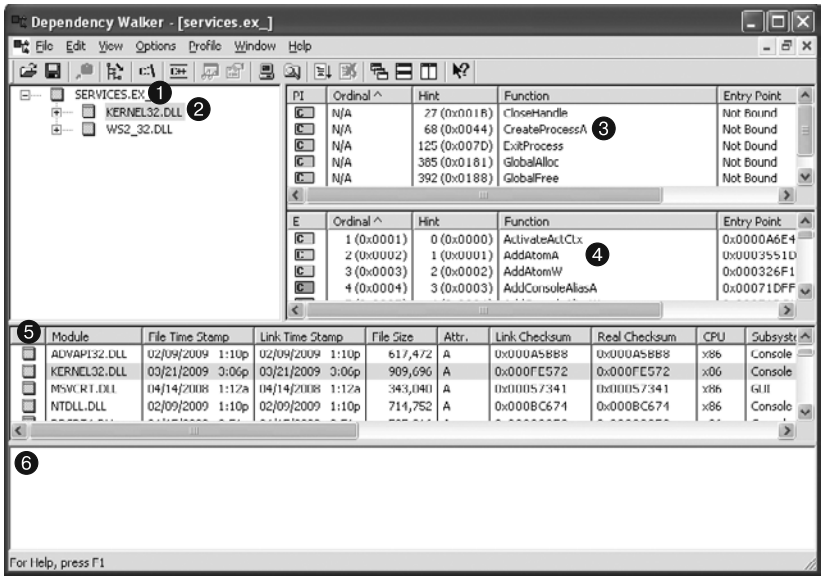

Рис. 1.6. Окно программы Dependency Walker

вы можете обратиться к панели ❹, чтобы сопоставить порядковый номер функции с ее названием.

На двух нижних панелях, ❺ и ❻, выводятся дополнительные сведения о версиях DLL, загружаемых при запуске программы, и отображаются ошибки.

Информация о том, какие DLL использует программа, может многое сказать о ее возможностях. В табл. 1.1 перечислены и описаны популярные DLL.

Таблица 1.1. Популярные DLL

DLL	Описание
Kernel32.dll	Очень распространенный DLL-файл, содержащий базовые функции: доступ и управление памятью, файлами и устройствами
Advapi32.dll	Обеспечивает доступ к ключевым компонентам Windows, таким как Диспетчер служб и Реестр
User32.dll	Содержит компоненты пользовательского интерфейса, такие как кнопки, полосы прокрутки и элементы для взаимодействия с пользователем
Gdi32.dll	В этом файле находятся функции для выполнения графических операций и графического вывода
Ntdll.dll	Интерфейс к ядру Windows. Исполняемые файлы обычно не импортируют его напрямую, но он всегда импортируется внутри Kernel32.dll. Если он импортирован напрямую, это означает, что автор программы намеревается использовать возможности, не свойственные обычным приложениям Windows. Этот интерфейс применяется для таких задач, как скрытие функциональности или манипуляция процессами

DLL	Описание
WSock32.dll и Ws2_32.dll	Это сетевые DLL. Программа, которая обращается к этим файлам, скорее всего, подключается к сети или выполняет какие-то сетевые задачи
Wininet.dll	Этот файл содержит высокоуровневые сетевые функции, реализующие такие протоколы, как FTP, HTTP и NTP

Соглашения об именовании функций

При оценке незнакомых Windows-функций стоит помнить о нескольких соглашениях об именовании, чтобы избежать путаницы. Например, вам часто будут попадаться функции с суффиксом Ex, такие как CreateWindowEx. Когда компания Microsoft выпускает несовместимые обновления, поддержка старых функций обычно продолжается. Новым функциям присваиваются те же имена, но с добавлением суффикса Ex. Если функция претерпела два значительных обновления, она может содержать в своем имени два таких суффикса.

Многие функции, принимающие строки в качестве параметров, имеют в конце своих названий A или W: например, CreateDirectoryW. Эти буквы не упоминаются в документации — они просто указывают на то, что функция принимает строковый параметр и имеет две версии: одна для строк в формате ASCII, а другая — для широкосимвольных строк. При поиске таких функций в официальной документации не забывайте убирать A или W в конце.

Импортированные функции

Заголовок PE-файла также содержит информацию о конкретных функциях, используемых программой. Их названия могут дать представление о том, что делает исполняемый файл. Компания Microsoft предоставляет отличную документацию для Windows API в своей онлайн-библиотеке Microsoft Developer Network (MSDN). В приложении A вы найдете список функций, которые часто используются вредоносными программами.

Экспортированные функции

DLL- и EXE-файлы могут экспортировать свои функции, чтобы взаимодействовать с другими программами и кодом. Обычно в DLL реализована одна или несколько функций, экспортирующихся для использования в любом исполняемом файле, который пожелает их импортировать.

PE-файл содержит информацию о том, какие функции экспортируются программой. Поскольку библиотеки DLL специально созданы для предоставления своей функциональности исполняемым файлам, экспортируемые функции обычно встречаются именно в них. EXE-файлы не предназначены для того, чтобы делиться

своими возможностями с другими программами, поэтому экспортируемые функции в них являются редкостью и обычно несут в себе полезную информацию.

Во многих случаях разработчики ПО дают своим экспортируемым функциям меткие имена. Традиционно названия берутся из документации Microsoft. Например, чтобы запустить программу в качестве службы, вам сначала нужно определить функцию `ServiceMain`. Наличие экспортируемой функции с этим именем означает, что вредоносная программа выполняется в рамках службы.

Хотя компания Microsoft использует название `ServiceMain` и разработчики обычно следуют ее примеру, данная функция может называться как угодно. В связи с этим названия экспортируемых функций не имеют особого значения при работе со сложными вредоносными программами. Если вредонос и экспортирует какие-то функции, он, скорее всего, либо вообще не станет их никак называть, либо даст им имена, которые могут лишь ввести в заблуждение.

Для просмотра сведений об экспорте можно использовать программу Dependency Walker, описанную чуть выше, в разделе «Исследование динамически скомпонованных функций с помощью Dependency Walker». Чтобы получить список экспортируемых функций, щелкните на имени файла, который вас интересует. Окно ❹ на рис. 1.6 выводит все функции, экспортированные файлом.

Статический анализ на практике

Теперь, когда вы получили представление об основах статического анализа, пришло время рассмотреть реальное вредоносное ПО. Мы исследуем предполагаемый кейлогер и упакованную программу.

PotentialKeylogger.exe: неупакованный исполняемый файл

В табл. 1.2 приводится краткий список функций, импортированных файлом `PotentialKeylogger.exe` (как показала Dependency Walker). Столь большое количество функций говорит о том, что файл не упакован.

Как и большинство программ среднего размера, этот исполняемый файл импортирует множество функций. Лишь небольшая часть из них полезна с точки зрения анализа безопасности. В этой книге мы будем время от времени возвращаться к импорту во вредоносном ПО, заостряя внимание на самых актуальных для нас функциях.

Если вы не уверены в назначении функции, нужно поискать ее описание. Чтобы помочь вам в этом, в приложении А мы перечислили множество функций, представляющих наибольший интерес при анализе вредоносного кода. Если не найдете ее там, попробуйте поискать в MSDN.

Будучи новичком, вы потратите много времени на поиск функций, не играющих особой роли, но вы быстро научитесь распознавать, какие из них имеют значение, а какие — нет. В этом примере мы покажем вам множество импортируемых функций,

которыми можно пренебречь, чтобы вы начали привыкать к исследованию больших объемов данных и поиску ключевых фрагментов.

Таблица 1.2. Краткий список DLL и функций, импортированных файлом PotentialKeylogger.exe

Kernel32.dll	**User32.dll**	**User32.dll (продолжение)**
CreateDirectoryW	BeginDeferWindowPos	**ShowWindow**
CreateFileW	CallNextHookEx	ToUnicodeEx
CreateThread	CreateDialogParamW	TrackPopupMenu
DeleteFileW	CreateWindowExW	TrackPopupMenuEx
ExitProcess	DefWindowProcW	TranslateMessage
FindClose	DialogBoxParamW	UnhookWindowsHookEx
FindFirstFileW	EndDialog	UnregisterClassW
FindNextFileW	GetMessageW	UnregisterHotKey
GetCommandLineW	GetSystemMetrics	
GetCurrentProcess	GetWindowLongW	**GDI32.dll**
GetCurrentThread	GetWindowRect	GetStockObject
GetFileSize	GetWindowTextW	SetBkMode
GetModuleHandleW	InvalidateRect	SetTextColor
GetProcessHeap	IsDlgButtonChecked	
GetShortPathNameW	IsWindowEnabled	**Shell32.dll**
HeapAlloc	LoadCursorW	CommandLineToArgvW
HeapFree	LoadIconW	SHChangeNotify
IsDebuggerPresent	LoadMenuW	SHGetFolderPathW
MapViewOfFile	MapVirtualKeyW	ShellExecuteExW
OpenProcess	MapWindowPoints	ShellExecuteW
ReadFile	MessageBoxW	
SetFilePointer	**RegisterClassExW**	**Advapi32.dll**
WriteFile	**RegisterHotKey**	RegCloseKey
	SendMessageA	RegDeleteValueW
	SetClipboardData	RegOpenCurrentUser
	SetDlgItemTextW	RegOpenKeyExW
	SetWindowTextW	RegQueryValueExW
	SetWindowsHookExW	RegSetValueExW

В обычной ситуации мы не знаем наперед, что эта вредоносная программа может быть кейлогером. Чтобы получить о ней какие-то сведения, нам бы пришлось просмотреть ее функции. Нас интересуют только те функции, которые проливают свет на возможности программы.