

*В. А. Букасов*
Московский инженерно-физический институт (государственный университет)

# ПРОВЕДЕНИЕ АВТОМАТИЗИРОВАННОГО АУДИТА ЗАЩИЩЕННОСТИ ПО

*В данной статье описывается методика динамического автоматизированного аудита внешних защит, также называемых навесными защитами, для программного обеспечения. Автор предлагает описание разработанной автоматизированной системы по снятию навесных защит с широким набором возможностей.*

В последнее время одним из наиболее часто используемых методов для защиты программного обеспечения является использование внешних, или навесных, защит: упаковщиков и протекторов. Это объясняется тем, что они разработаны специально для защиты программ и, как правило, содержат в себе большинство основных программных методов защиты. Они отличаются легкостью применения: достаточно лишь указать сам файл для защиты и основные параметры защиты, и протектор сам защитит файл с использованием выбранных настроек. Упакованные программы невозможно исследовать в статике, динамический же анализ также значительно усложняется. Но тенденция применять протекторы распространяется не только на защиту приложений: уже больше половины появляющихся вредоносных программ защищены каким-нибудь упаковщиком или протектором, что мешает их исследованию. Также большинство разработчиков программного обеспечения имеют весьма слабое представление о степени защищенности их программы от атак, полагаясь на какой-нибудь протектор, в результате чего их продукт быстро оказывается взломан, а они даже не подозревают, что же было сделано не так.

В данной статье описывается разработанный автором программный комплекс [1, 2], призванный распаковывать исполняемые файлы и динамически загружаемые библиотеки, упакованные упаковщиками и протекторами, значительно упрощая динамический анализ программ и делая возможным проведение статического анализа, что полезно при исследовании вредоносных программ и помогает разработчикам оценить стойкость выбранного протектора к атакам аналитика. Существуют и аналогичные работы [3, 4], но они не лишены недостатков, в частности, отладочное ядро уязвимо ко многим антиотладочным приемам, и комплексы оказываются не в состоянии распаковать относительно сложные протекторы.

Принцип работы упаковщиков состоит в следующем. Когда происходит загрузка на исполнение неупакованного файла, загрузчик операционной системы Windows выполняет следующие основные действия: создает отдельное адресное пространство, располагает в этом адресном пространстве образ запущенного файла, вносит изменения в этот образ, заполняя таблицу импорта и используя данные из таблицы перемещаемых элементов, и передает управление на

точку входа (она же оригинальная точка входа). При обработке файла упаковщиком код и данные в программе шифруются для усложнения распаковки и иногда сжимаются для уменьшения размера, а в конец файла добавляется дополнительная секция, содержащая код протектора (так называемая заглушка), который как раз и занимается расшифровкой файла и выполнением некоторой работы загрузчика операционной системы Windows, причем новая точка входа устанавливается как раз на добавленную заглушку. На заглушку возложены следующие основные функции: расшифровать код и данные, заполнить таблицу импорта, внести изменения в образ, основываясь на данных из таблицы перемещаемых элементов, и передать управление на оригинальную точку входа для начала исполнения программы. Все описанные функции должна выполнять именно заглушка, а не загрузчик операционной системы Windows, поскольку при работе загрузчика код и данные находятся в зашифрованном и, возможно, сжатом виде, при внесении в них изменений последующая расшифровка пройдет некорректно. Таким образом, код и данные программы в файле на диске лежат в зашифрованном и, возможно, сжатом виде, поэтому без распаковки провести статический анализ такой программы невозможно.

Снятие упаковщика или протектора обычно сводится к нескольким основным шагам.

*Нахождение оригинальной точки входа*. Когда начинает выполняться код, содержащийся по адресу оригинальной точки входа, код и данные программы уже должны находиться в распакованном и расшифрованном виде, поэтому нахождение этого адреса необходимо для выполнения остальных шагов и получения корректно работающей программы после распаковки.

*Считывание программы из памяти с оригинальной точки входа*. Когда код и данные программы уже расшифрованы, можно забрать их из памяти и сохранить в файл на жесткий диск.

*Восстановление импорта*. Таблица импорта заполняется динамически для каждой загружаемой программы при каждом ее запуске. Это связано с тем, что адреса одних и тех же функций импорта для разных версий операционной системы могут быть разными. Поэтому образ без корректно восстановленной таблицы импорта, скорее всего, не будет работать на другой версии операционной системы.

*Восстановление таблицы перемещаемых элементов (только для библиотек)*. Таблица перемещаемых элементов нужна для того, чтобы вносить изменения в загруженный образ, если образ был загружен по адресу, отличному от ожидаемого адреса. Для загружаемых файлов таблица перемещаемых элементов не является необходимой, поскольку образ файла первым записывается в адресное пространство, когда оно еще пустое. А вот библиотеки загружаются, когда адресное пространство может быть занято самим загружаемым файлом или другими библиотеками. Если в библиотеке не восстановить таблицу перемещаемых элементов, библиотека не будет корректно работать, если будет загружена по адресу, отличному от адреса, по которому она была загружена при распаковке.

Снятие протектора обычно дело нетривиальное и требует определенных знаний и умений, поскольку в нем используется множество различных способов защиты. В основном это похоже и на снятие упаковщиков — необходимо проделать те же самые шаги, но протекторы пытаются противодействовать выполнению некоторых шагов, и порой приходится применять нестандартные методы.

Исходя из описанных выше шагов, система для распаковки может состоять из следующих основных модулей.

*Отладочное ядро*. Отладочное ядро может состоять из двух частей: драйвера и пользовательской части. Оно предназначено для обработки событий срабатывания точек останова и работы с контекстом. Использование драйвера в отладочном ядре позволяет обходить многие антиотладочные приемы, что не может быть реализовано без его использования, и присоединяться к уже запущенным процессам для распаковки.

*Модули определения упаковщика и протектора*. Эти модули носят скорее косметический характер — их работа не является необходимой, они призваны только информировать пользователя о предположительно обнаруженном упаковщике или протекторе, чтобы в случае неудачной распаковки пользователь мог обратиться к соответствующим статьям и распаковать файлы вручную. Определение названия и версии навесной защиты проще всего реализовать через сигнатуры, как реализовано в [5, 6].

*Модули определения оригинальной точки входа*. Способов реализовать данные модули может быть несколько. Так как оригинальная точка входа расположена обычно в первой секции, следить на предмет выполнения можно за кодом из первой секции. Реализовать это можно с использованием драйвера [7], помечая страницы памяти как отсутствующие, либо на пользовательском уровне поменять атрибуты страниц памяти [8]. Поскольку многие упаковщики стараются сохранить первоначальное состояние стека на момент вызова оригинальной точки входа, можно контролировать обращения к стеку. Либо можно использовать сигнатурный метод и искать нужное место по сигнатурам, но этот метод часто будет давать ошибки.

*Модуль восстановления импорта*. Модуль восстановления импорта реализует нахождение и восстановление таблицы импорта для файла. Для восстановления импорта можно воспользоваться разными методами. Первый из вариантов основан на статическом поиске переходников на функции импорта. Данный метод восстановления импорта просто проходит по всему образу файла в памяти и ищет переходники, похожие на вызов функций импорта. К преимуществам метода можно отнести то, что трассировка функций не проводится. Из недостатков стоит отметить, что метод не будет работать, если переходники ведут не непосредственно на функцию импорта, а на переходник протектора, который сам позже вызовет нужную функцию (так называемый перенаправленный импорт). Можно использовать и другой метод, в отличие от прошлого метода, он работает уже не в статике, а в динамике. Сначала ищутся переходники, похожие на вызов функций импорта, а затем запускается трассировка этих переходников. Данный метод имеет ряд преимуществ, он позволяет обрабатывать даже довольно сложные схемы перенаправления импорта. Но метод не лишен и недостатков, порой программа может аварийно завершиться, сделав дальнейшую работу по трассировке импорта и распаковке невозможной. И можно еще использовать третий метод. Его суть заключается в том, что полностью таблица импорта не восстанавливается, а в новую таблицу импорта записывается лишь по одной первой функции из каждой библиотеки, подгруженной в адресное пространство распаковываемого процесса. Это нужно для того, чтобы при загрузке на исполнение распакованного файла загрузчик операционной системы Windows загрузил все необходимые библиотеки. Ставка здесь делается на то, что адреса самих функций импорта либо считаются динамически в процессе исполнения, либо не изменялись в системе с момента распаковки приложения. Метод находит свое применение в том случае, если перенаправление импорта слишком сложное и предыдущими методами его восстановить не представляется возможным. Но стоит иметь в виду, что при изменении адресов функций импорта (что может быть в другой версии операционной системы) распакованное приложение может работать некорректно, поскольку таблица импорта динамически не заполняется всеми используемыми функциями на этапе загрузки. Тем не менее зачастую этого метода вполне достаточно при исследовании вредоносных программ.

Следует помнить, что поиск переходников на функции импорта необходимо осуществлять не только по стандартным переходникам вида call dword ptr[xxx]/jmp dword ptr[xxx], но также и по обращениям через регистры, характерным для программ, скомпилированных из языка C/C++, а также не забывать о возможности поиска и по обращениям вида call xxx/jmp xxx, как порой перенаправляют некоторые протекторы.

Помимо описанных выше возможностей в систему могут быть включены некоторые дополнительные полезные возможности, расширяющие ее функционал.

*Подсчет остановов на оригинальной точке входа*. Когда программа запускается, ожидается срабатывание точки останова на оригинальной точке входа. Но может быть так, что код по адресу оригинальной точки входа исполняется несколько раз, а лишь затем уже происходит нужное выполнение. Данная проблема решается путем запуска программы, подсчета числа исполнений кода на оригинальной точке входа и последующего останова на последнем исполнении.

*Восстановление таблицы перемещаемых элементов (только для библиотек)*. Для восстановления перемещаемых элементов в системе можно использовать следующий метод: одна и та же библиотека распаковывается дважды, загружаясь при этом по разным адресам. В результате две распакованные библиотеки сравниваются, в них ищутся различия и на основании этих различий и формируется новая таблица перемещаемых элементов.

*Удаление лишних секций и восстановление ресурсов*. После того как файл уже распакован, необходимость в секциях протектора отпадает, поэтому для уменьшения размера файла их можно удалить. Поскольку секция ресурсов обычно идет последней в незапакованном файле, то всё, что после секции ресурсов, обычно принадлежит упаковщику или протектору. Поэтому в системе можно реализовать удаление секций после ресурсов. Но при этом некоторые протекторы забирают часть ресурсов из секции ресурсов в свою секцию, поэтому необходимо не забыть их перестроить, чтобы программы [9] могли их корректно обрабатывать.

*Сохранение оверлея*. В некоторых программах порой встречается оверлей, это данные в конце файла, которые не входят в образ файла. В оверлее некоторые троянские программы хранят свои настройки, в нем могут быть и какие-нибудь дополнительные данные. Оверлей не попадает в образ файла, поскольку он не входит в сам образ файла в адресном пространстве процесса, поэтому его необходимо восстанавливать отдельно. Восстановление оверлея является довольно простой задачей — нужно всего лишь взять из исходного упакованного файла все данные, не входящие в образ, и добавить их в конец распакованного файла.

*Использование сценариев автоматизации*. Поскольку система разрабатывается для общего случая и не была ориентирована на работу с какими-либо конкретными протекторами, с некоторыми приемами по усложнению распаковки она справиться не может. Для того чтобы побороть такие сложности, можно создать поддержку сценариев автоматизации, или скриптов [10], которая позволяет любому пользователю написать собственный скрипт и уже с его помощью распаковывать даже сложные протекторы. В скриптах можно контролировать основные переменные системы и вызывать множество различных функций. С помощью скриптов можно не только проходить до оригинальной точки входа, но и полностью контролировать процесс распаковки и вносить изменения в образ файла. Скрипты значительно расширяют возможности системы и позволяют распаковывать даже сложные протекторы.

## СПИСОК ЛИТЕРАТУРЫ:

1. *Букасов В. А.* Автоматизация анализа стойкости навесных защит // Технологии Microsoft в теории и практике программирования: труды V Всероссийской конференции студентов, аспирантов и молодых ученых. М.: Вузовская книга, 2008. С. 131–133.

2. *Букасов В. А., Краснопевцев А. А.* Автоматизация динамической распаковки программ // Информационная безопасность: материалы X Международной научно-практической конференции. Ч. 2. Таганрог: Изд-во ТТИ ЮФУ, 2008. С. 13–15.

3. *Royal P., Halpin M., Dagon D., Edmonds R., Lee W.* PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. College of Computing, Georgia Institute of Technology, 2006.

4. *Sun L., Ebringer T., Boztas S.* Hump-and-dump: efficient generic unpacking using an ordered address execution histogram. Department of Computer Science and Software Engineering, The University of Melbourne, Australia, 2008.
5. Программа для определения навесной защиты PEId. URL: http://cracklab.ru/download.php?action=get&n=NTYw.
6. Модуль для определения навесной защиты PESniffer из состава PETools. URL: http://cracklab.ru/download.php?action=get&n=MTU1.
7. Модуль для нахождения оригинальной точки входа Dream Of Every Reverser. URL: http://deroko.phearless.org/doer.html.
8. Модуль для нахождения оригинальной точки входа OEPFinder by Human. URL: http://www.exetools.com/forum/printthread.php?t=8841.
9. Программа для редактирования ресурсов Restorator. URL: http://www.bome.com
10. Язык для скриптов Lua. URL: http://www.lua.org.

*С. В. Запечников (к. т. н., доцент)*
Московский инженерно-физический институт (государственный университет)

# КОНТРОЛЬ ЦЕЛОСТНОСТИ ФУНКЦИОНАЛЬНЫХ РЕСУРСОВ СРЕДСТВ ЗАЩИТЫ ИНФОРМАЦИИ В РАСПРЕДЕЛЕННОЙ СРЕДЕ

*В работе решается задача контроля функциональных ресурсов средств защиты информации (СЗИ). К контролируемым ресурсам относятся фрагменты файловой системы (ФС) и баз данных (БД), а также области оперативной и программируемой памяти, в которых содержатся объекты, реали-зующие функциональность СЗИ. Принципы решения задачи основаны на самоконтроле и взаимном контроле целостности выделенного множества объектов узлами распределенной компьютерной системы (РКС), функционирующими в составе СЗИ.*

## Введение

Корректная реализация протоколов и алгоритмов, обеспечивающих безопасность информации в компьютерных системах (КС), возможна лишь при условии обеспечения целостности самих ресурсов КС, ответственных за выполнение этих действий. Несмотря на наличие целого ряда программно-аппаратных средств контроля целостности ПО общего назначения, эта задача по-прежнему остается актуальной именно для СЗИ. Поскольку сами средства контроля целостности ресурсов с полным основанием могут быть отнесены к СЗИ, в решении этой задачи возникает специфика, связанная по сути с необходимостью обеспечивать самоконтроль и собственную безопасность СЗИ. Частным случаем рассматриваемой задачи является обеспечение целостности ресурсов средств криптографической защиты информации и средств управления криптографическими ключами, которые должны сохранять стойкость в условиях частичного разрушения и (или) компрометации системы.

Под функциональными ресурсами СЗИ будем понимать любые объекты КС, в которых содержатся объекты, реализующие те или иные функции СЗИ. Понятие «функциональные ресурсы» в рамках настоящей работы используется в противоположность понятию «информационные ресурсы», обозначающему данные, над которыми или с помощью которых СЗИ выполняют свои функции. Методы контроля целостности информационных ресурсов достаточно традиционны и хорошо разработаны: к ним, прежде всего, относятся коды, обнаруживающие и исправляющие ошибки, функции хэширования. Информационные ресурсы выступают для средств контроля целостности абсолютно пассивными и независимыми друг от друга объектами. Напротив, функциональные ресурсы организованы в систему, их число может быть весьма значительно, они распределены по множеству узлов РКС, а дополнительную сложность в их контроль вносит тот факт, что при выполнении процедур контроля они должны продолжать выполнять свои основные функции.