УДК 004.056

К.Ю. Попырко, О.Г. Шевченко
Донецкий национальный технический университет, г. Донецк
кафедра компьютерной инженерии

СПОСОБЫ ЗАЩИТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ОТ НЕСАНКЦИОНИРОВАННОЙ МОДИФИКАЦИИ

Аннотация

Попырко К.Ю., Шевченко О. Г Способы защиты программного обеспечения от несанкционированной модификации. *В статье дан обзор механизмов зашиты, применяемых к программному обеспечению с целью усложнения процесса его изменения. Представлены функциональное описание и характеристики популярных программ-протекторов.*

Актуальность темы. Проблема защиты программного обеспечения (ПО) от несанкционированной модификации остается актуальной для большинства программистов, которые занимаются разработкой платного программного обеспечения. Хорошо защищенная программа гораздо дольше остается не взломанной, что способствует получению максимальной прибыли от её продаж.

Цель статьи — анализ методов взлома программного обеспечения.

Введение. Взлом защиты программного обеспечения невозможен без изучения защитных механизмов, используемых в этом ПО. Изучение программного продукта может осуществляться методом статического и/или динамического анализа. При статическом анализе разработка алгоритма взлома защиты производится на основе анализа результатов дизассемблирования или декомпиляции исследуемой программы. Для динамического анализа взламываемую программу запускают в среде отладчика, при этом становится возможным контроль всех изменений, возникающих в процессе функционирования приложения. В процессе динамического анализа, используя режим отладки, производят пошаговое прохождение "защитных" алгоритмов программы, например алгоритмов проверки или генерации корректного регистрационного кода.

Схема получения исходного кода на языке ассемблера для программного обеспечения представлена на рисунке 1.

Рисунок 1 — Схема получения исходного кода на языке ассемблера для программного обеспечения

Имея доступ к исходному коду, исследователь может изменить исполняемый файл таким образом, чтобы изменить поведение программного обеспечения на поведение, соответствующее зарегистрированной версии программы. Исходя из вышеизложенного, можно сказать, что для защиты программного обеспечения от несанкционированного исследования и/или модификации, необходимо максимально усложнить статический и динамический анализ.

Способы защиты. Среди наиболее часто применяемых методов защиты ПО можно выделить следующие:

1. Упаковка и/или шифрование исходного файла

Защищаемый файл полностью или частично упаковывается и/или шифруется криптографическим алгоритмом. Распаковка и/или расшифровка, а также передача управления на оригинальный код происходит в памяти, что делает невозможным проведение статического анализа. Однако, в процессе динамического анализа такая защита может быть полностью снята исследователем, т.е. после распаковки и/или расшифровки исходный файл находится в памяти компьютера в «чистом» виде. Т. о., после снятия дампа с области памяти, в которой находится исходный файл, можно продолжить анализ.

2. Запутывание (обфускация) кода

Эффективность данного метода достигается за счет использования особенностей человеческого фактора - чем сложнее исходный код и чем больше ресурсов использует приложение, тем сложнее исследователю понять логику работы программы, а, следовательно, и взломать примененные средства защиты. Обфускация (запутывание) кода может быть произведена различными способами, при этом наибольший эффект защиты достигается при комбинировании их друг с другом:

1. Вставка мусорного кода — вставка случайного числа инструкций, которые не изменяют поведение программы, но при этом вносят неясность в

код, увеличивая тем самым количество необходимого времени, требуемого исследователю на изучение алгоритма.

2. Изменение потока выполнения — вставка случайных условных и/или безусловных переходов в код, которые запутывают код, и, соответственно, затрудняют понимание алгоритма исследователем.

3. Мутация кода — одно- или многократные преобразования существующих инструкций в другие, не изменяющие поведение программы.

4. Виртуализация кода — преобразование кода в код абстрактной виртуальной машины, код которой располагается в исполняемом файле, для исключения возможности понимания исходного алгоритма в ходе статического анализа.

3. Скрытие вызовов API-функций

Позволяет скрыть от исследователя список API, которые использует защищаемая программа. Скрытие может осуществляться несколькими способами:

1. Скрываемые функции отсутствуют в таблице импорта, адреса этих функций определяются динамически уже во время работы защищенного файла, а не на этапе работы стандартного загрузчика Windows.

2. Эмуляция некоторых API-функций — перенаправление прямого вызова API-функции на вызов внутренней функции, которая полностью повторяет поведение скрываемой функции.

4. Применение анти-отладочных методов

Способ заключается в определении, запущена ли программа в среде отладчика — если да, выполнение программы завершается. Наиболее распространенные методы:

1. Использование API-функций (IsDebuggerPresent, CheckRemoteDebuggerPresent).

2. Поиск отладчика в системе (по имени процесса, по имени/классу окна отладчика, по создаваемым объектам ядра).

Обзор существующих решений защиты программ от взлома. Основным инструментом, применяемым для защиты приложений от взлома, являются программы-протекторы. Описание и характеристики наиболее популярных из них:

1. VMProtect — средство защиты программного обеспечения. Поддерживаемые форматы файлов: EXE, DLL; поддерживаемые операционные системы: все 32/64-разрядные операционные системы семейства Windows. Используемые методы защиты:

1) Обфускация кода в VMProtect: виртуализация, мутация и смешанный метод защиты, сочетающий мутацию кода приложения с его последующей виртуализацией. При виртуализации участков кода, VMProtect позволяет использовать несколько отличных друг от друга виртуальных машин для защиты разных участков кода одного приложения, что еще больше

усложняет процесс взлома защиты, так как взломщику будет необходимо анализировать архитектуру уже нескольких виртуальных машин. В процессе мутации, помимо замены команд их функциональными аналогами, в код приложения добавляются «мусорные» команды, и случайные условные и безусловные переходы.

2) Применение анти-отладочных приемов для защиты от динамического анализа — в VMProtect применяются различные методы обнаружения отладчика.

3) Скрытие вызовов API-функций.

2. Enigma Protector — система защиты исполняемых файлов. Содержит широкий ряд уникальных функций и возможностей, основной целью которых является защита исполняемых файлов от нелегального копирования, взлома, модификации и исследования. Поддерживаемые форматы файлов: x86/x64 EXE/DLL файлы для любых версий семейства ОС Windows. Используемые методы защиты:

1) Обфускация кода выполняется виртуализацией заданных участков кода.

2) Применение различных анти-отладочных приемов

3) Скрытие вызовов API-функций.

3. Themida — система защиты исполняемых файлов. Поддерживаемые форматы файлов: x86/x64 EXE и DLL исполняемые файлы семейства операционных систем Windows. Для защиты используется технология «SecureEngine», которая использует следующие техники защиты:

1) Для обфускации кода, применяются следующие техники: SmartMetamorph и MutatorEngine (мутация кода), GarbageCode (вставка мусорного кода), VirtualMachine (виртуализация кода).

2) Шифрование части исполняемого файла: техника CodeEncrypt (зашифрованные блоки кода расшифровываются только при обращении к ним).

3) Применение анти-отладочных приемов — MemoryGuard (защита памяти приложения от изменения в момент выполнения), DebuggerGuard (техника обнаружения отладчиков в системе), AntiBreakpoints (техника, затрудняющая установку точек останова при запуске приложения в среде отладчика), AntiDumperPro (техники, затрудняющие получение «дампа» процесса защищенного приложения).

4) Скрытие вызовов API-функций — SecureAPIWrapper (скрытие таблицы импорта).

4. Obsidium — система защиты программного обеспечения от несанкционированного исследования и модификации. Поддерживаемые форматы файлов: x86/x64 EXE и DLL исполняемые файлы семейства ОС Windows. Используемые методы защиты:

1) Для обфускации кода применяются упаковка, шифрование и виртуализация кода.

2) Применение анти-отладочных приемов.
3) Скрытие API-функций.

Сравнение рассмотренных продуктов. На рисунках 2, 3, 4 приведены сравнительные диаграммы количественных параметров рассмотренных протекторов, полученные при защите тестового приложения.

Тестовое приложение представляет собой простейший аналог коммерческого приложения с защитой от несанкционированного копирования с помощью лицензионного ключа. Алгоритм программы не линеен, т. к. существует условие проверки лицензионного ключа — в случае, если значение лицензионного ключа совпадает с заданным в программе, на экран выводится сообщение «OK! Program registered!», иначе — «Error! Invalid serial number!». Для чтения из файла используются стандартные функции runtime-библиотеки C++ “fopen”, “fread” и “fclose”. Для преобразования текстового представления лицензионного ключа в численное, используется функция той же библиотеки “atol”.

Параметры защиты, установленные в каждом из рассматриваемых протекторов:

– Обфускация кода: виртуализация одной выбранной функции
– Скрытие вызовов API-функций
– Анти-отладочные приемы: обнаружение отладчиков

Все остальные параметры каждого из протектора были оставлены в их значении «по умолчанию».

Исходные значения сравниваемых количественных параметров:

– размер исходного файла: 2 (кБ);
– объем оперативной памяти, занимаемой программой: 2076 (кБ);
– время выполнения программы: 0.01 (сек).

Сравнительная характеристика описанных протекторов по параметру «размер файла» после обработки протектором приведена на рисунке 2.

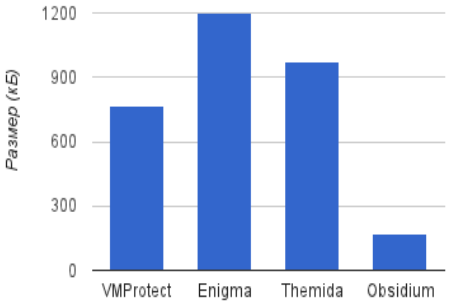

Рисунок 2 — Сравнительная характеристика описанных протекторов по параметру «размер файла»

Сравнительная характеристика описанных протекторов по параметру «время выполнения программы» после обработки протектором приведена на рисунке 3.

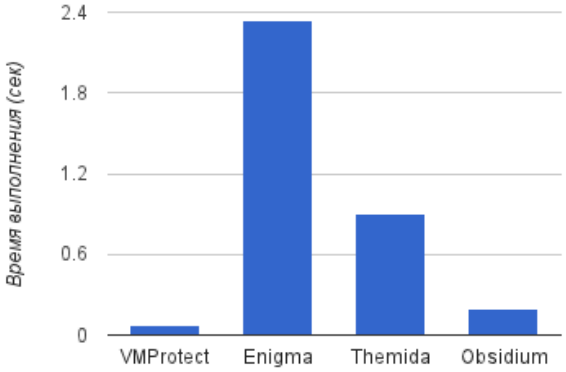

Рисунок 3 — Сравнительная характеристика описанных протекторов по параметру «время выполнения программы»

Сравнительная характеристика описанных протекторов по параметру «объем занимаемой оперативной памяти» после обработки протектором приведена на рисунке 3.

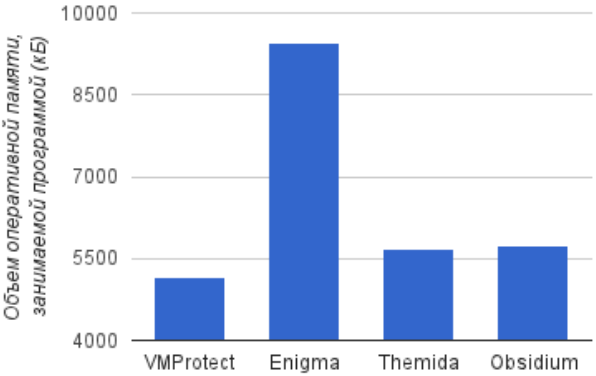

Рисунок 4 — Сравнительная характеристика описанных протекторов по параметру «объем занимаемой оперативной памяти»

Исходя из вышеизложенного, можно сказать что защита, добавленная протектором Enigma, более «сложная», т. к. защищенный файл имеет

максимальный размер, что означает большее количество добавленного кода, который необходимо будет проанализировать исследователю. Также, на выполнение кода ушло максимальное количество времени. Однако, при более глубоком анализе защищенного приложения в отладчике, было выяснено, что значительное увеличение времени выполнения достигается благодаря добавлению большого количества бесполезных циклов. Т. о., необходимо отметить, что данный пункт не может рассматриваться как пункт, однозначно оценивающий степень защищенности приложения. Также, нужно отметить, что выбранные количественные параметры, по которым сравнивались рассматриваемые протекторы, не позволяют в полной мере оценить сложность взлома программы, защищенной соответствующим протектором, т.к. важнейшим фактором является уровень квалификации исследователя.

Выводы:

В данной работе были рассмотрены методы взлома программного обеспечения, а также методы, затрудняющие исследование и взлом программного обеспечения. Также был проведен сравнительный анализ наиболее популярных протекторов, в которых применяются рассмотренные методы защиты.

Список литературы

1. Казарин О.В. Безопасность программного обеспечения компьютерных систем. – М.: МГУЛ, 2003.

2. Касперски К. Техника отладки программ без исходных текстов. – СПб.: БХВ-Петербург, 2005.

3. WASM [Electronic resource] / Интернет-ресурс. – Режим доступа : www/ URL: https://wasm.ru/forum/

4. Chris Eagle. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. No Starch Press, Inc., 2008.

5. Randall Hyde. The Art of Assembly Language, 2nd Edition. No Starch Press, Inc., 2010.

6. Michael Sikorski, Andrew Honig. Practical Malware Analysis, The Hands-On Guide to Dissecting Malicious Software. No Starch Press, Inc., 2012.