УДК 004.056.57

А.С. Петров, А.А. Петров

Восточноукраинский национальный университет им. Владимира Даля, Луганск

МЕТОДЫ ЗАЩИТЫ ПРОГРАММНОГО КОДА

Рассмотрены вопросы построения системы защиты программного кода. Описаны методики нарушения целостности исполняемого программного кода и способы их устранения. Рассмотрено применение технологии виртуальных машин в контексте защиты исполняемого кода от анализа. Дано обоснование применения виртуальной машины как оптимального способа противодействия анализу исполняемого кода.

Ключевые слова: *программный код, защита исполняемого кода, виртуальная машина, интерпретация, дизассемблирование.*

Введение

В настоящее время является достаточно актуальной (и вряд ли утратит актуальность) проблема обработки информации в компьютерных системах и сетях. В частности, до сих пор не решена задача защиты программного кода (ПК) и данных, которые обрабатываются ПК. Основными целями защиты кода программы являются:

1. Недопущение несанкционированного изменения алгоритма поведения программы.
2. Защита обрабатываемых данных от нелегитимного пользователя.
3. Обеспечение защиты программного обеспечения (ПО) от несанкционированного копирования.

Идеальной защиты в области программного обеспечения до сих пор не создано, хотя уже десятилетия идет война создателей защит и хакеров. Неизменно победителями выходят последние, лишь иногда оставляя пальму первенства создателям защит. Под защитой авторы понимают такой блок исполняемого кода, который противодействует (успешно или нет) нелегальному использованию программы. Это может быть защита от копирования с носителя информации, ввод серийного регистрационного номера программы, средство, ограничивающее максимальное количество лицензий в сети и так далее.

Далее рассмотрим методы и методики, с помощью которых достигаются описанные выше цели. Отметим, что ни одна из перечисленных целей на данный момент не достигнута и соответствующие им научно-технические задачи полностью не решены. Так как не имеет смысла изобретать защиту от неизвестной угрозы, не зная где, как, и когда будет произведена атака, будем рассматривать средства защиты, отталкиваясь от средств нападения. Другими словами, будем проводить анализ по пути от угрозы к средству защиты, выступив сначала на стороне злоумышленника, а затем на стороне специалиста по защите информации.

Актуальность проблемы защиты программного кода

В области изменение алгоритма поведения программы для злоумышленников есть огромное поле для деятельности. Дело в том, что подавляющая часть кода программ находится в открытом виде, т.е. имеют вид либо машинного кода, либо некоторого промежуточного байт-кода (программного ассемблера), выполняемого виртуальной машиной.

Основной проблемой, с которой сталкиваются программисты, создающие защищающий код (тот участок кода, который принимает решение, открыть или закрыть доступ к основному коду, иными словами, выполняет функции идентификации, аутентификации и авторизации), является проблема потенциальной возможности анализа и исправления данного двоичного кода. Причем, эта уязвимость, по мнению авторов, является одной из наиболее проблематичных в области создания защиты программного обеспечения.

На сегодняшний день широко известно уже не одно решений криптозащиты кода приложения, позволяющее обезопасить его от дизассемблирования, и привязать к аппаратному ключу, например, Guardant.

Возьмем наиболее популярный метод криптозащиты. Закодируем важные участки кода так, чтобы правильно декодировать их можно было только с помощью законно полученного ключа. Без ключа приложение перестает быть работоспособным, или работает в режиме оценочной версии. Алгоритм кодирования будет являться тестом законного использования. Условный переход IF «декодировано_правильно» THEN ... – это всего лишь формальность, которая исключит ошибку нарушения доступа при выполнении неверно декодированного участка. Приложение с криптозащитой остается уязвимым по отношению к атаке, когда взломщик покупает лицензированную копию и, запустив ее, снимает из дампа памяти декодированную версию. Следо-

вательно, сразу после исполнения декодированного участка приложения необходимо его вновь закодировать, или переносить обратно заранее сохраненный закодированный участок. У взломщика останется единственная возможность поймать момент, когда в памяти исполняется декодированный участок, сохранить рабочий код приложения и действовать таким образом далее, в конечном итоге собирая приложение в единое целое по кускам, но это ручная кропотливая работа, тем более если предусмотреть большое количество маленьких участков кода, с их последовательной расшифровкой. Борьба с атаками такого рода заключается в как можно большем количестве закодированных участков, их замене в каждой новой версии системы [1].

Для машинного кода создано достаточно дизассемблеров [2], способных представлять программу в удобном для просмотра виде (на языке ассемблер). Таким образом, изменив хотя бы одну машинную инструкцию, получаем такие возможные результаты:

- неработоспособный код;
- отложенный неработоспособный код (от нескольких секунд, до нескольких часов и дней);
- некорректно работающий код;
- намеренно некорректно работающий код.

Последствия таких событий очевидны. Причем наиболее плачевный результат ожидает нас в последнем случае. В зависимости от предметной области ПО, это может быть потеря крупных денежных сумм при расчетах, разрешение доступа к системе нелегитимному пользователю, и тому подобное – список можно продолжать бесконечно.

Обеспечение целостности кода и безопасности данных

Решений по обеспечению безопасности ПК существует несколько.

Во-первых, использовать упаковщики исполняемых файлов, например UPX [3]. Однако, это останавливает лишь неопытного взломщика. Сигнатура распаковщика, прикрепленного в начале файла, легко распознается как вручную, так и автоматически, после чего легко осуществляется обратный процесс приведения кода в изначальный неупакованный вид.

Во-вторых, часто применяются специальные средства защиты такие как, ASProtect - система программной защиты приложений от несанкционированного копирования, разработанная для быстрой установки функций защиты приложения и предназначенная специально для разработчиков ПО [4]. Чаще такая защита пишется разработчиком вручную, и как следствие, ее уровень оказывается еще хуже, т.к. для написания защиты требуется хотя бы минимальная квалификация в области написания защит ПО. Основные схемы таких средств можно увидеть на рис. 1.

Рис. 1. Схемы средств защиты исполняемого кода: А – защита вне программы; Б – защита внутри программы; В – защита рассеяна внутри программы

Основным неустранимым недостатком таких защит является уязвимое место соприкосновения основного кода с защитным кодом. Применяя обратный инжиниринг, в частности, дизассемблирование, с помощью дизассемблера, при котором прямой машинный код программы читается и понимается в своём чистом виде, только с помощью мнемоник машинного языка, становится возможным изменять, корректировать и даже удалять защитный код.

В результате простого анализа состояния дел в области защиты программного кода становится ясно, что все существующие направления защиты являются бесперспективными в плане достижения гарантированного результата.

Для преодоления таких трудностей автором предлагается принципиально новая методика, основанная на сочетании виртуальной машины и криптографии.

Виртуальная машина (англ. virtual machine) – это программная или аппаратная среда, исполняющая некоторый код (например, байт-код, шитый код, р-код или машинный код реального процессора), или спецификация такой системы [5].

Основная суть идеи – концепция виртуальной машины, исполняющей шитый код, в сочетании с

его динамическим шифрованием/дешифрованием на лету (рис. 2).

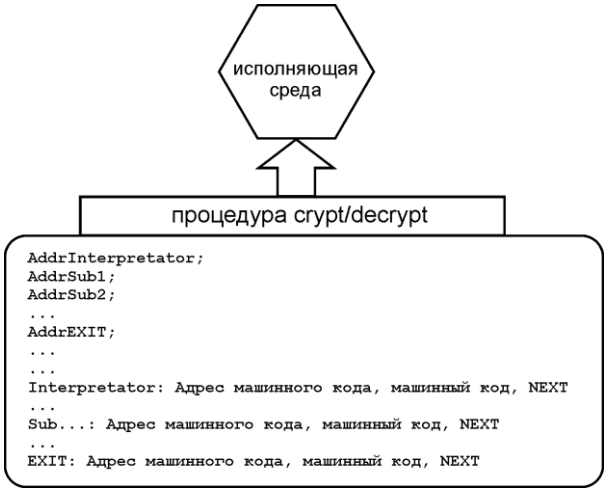

Рис. 2. Блок-схема виртуальной машины с динамическим шифрованием на лету

Шитый код - один из способов реализации промежуточной виртуальной машины при интерпретации языков программирования (наряду с байт-кодом).

Основное представление программы при использовании шитого кода — массив вызовов подпрограмм (рис. 2). Реализация шитого кода, способ хранения этих вызовов может быть различной. Этот код может обрабатываться интерпретатором (за которым утвердилось название адресный интерпретатор), или может быть простой последовательностью машинных инструкций вызова подпрограммы. Некоторый набор базовых подпрограмм виртуальной машины, использующей шитый код, реализуется в виде подпрограмм, написанных на обычном машинном коде [6].

Таким образом, сохраняется скорость выполнения кода программы, и, одновременно с этим, повышается его защищенность. Теоретически, выполняя дешифрование всего одной инструкции за «такт» позволяет обеспечить защиту программного кода с вероятностью p = 100%. Иными словами, в памяти компьютера в данный момент времени t_i будет находиться одна инструкция k_i. Это сделает реверсный инжиниринг кода невозможным.

Применить описанную технологию можно в различных вариациях. В качестве примера можно рассмотреть следующую последовательность действий:

1. В качестве ключа шифрования кода может выступать некоторая лицензия(ключевой файл), выданная пользователю (ограничивается несанкционированное копирование).

2. Весь защищаемый код программы содержит фрагменты защитного кода, рассеянный при помощи генератора псевдослучайной последовательности, который выполняемый ВМ, непосредственно внедренной в основную программу.

В результате таких действий оригинальный код программы оказывается «перемешанным» с защитным кодом, который, в свою очередь, представлен в виде зашифрованной последовательности данных, расшифровка которых для выполнения происходит по одной инструкции, в результате чего дизассемблировать защитный код не представляется возможным. При этом, хотя дизассемблирование машинного кода программы все еще доступно злоумышленнику, но не дает желаемого результата, т.к. отделение оригинального кода от внедренного защитного кода становится очень трудоемкой задачей, с не всегда предсказуемыми результатами. Для достижения еще более высокой степени защиты, возможно частичная реализация программы в виртуальной машине подсистемы защиты, например, взять для этого 5 – 10% кода от общего объема программы, занимающего 1 – 2% времени выполнения – это не даст проигрыш по скорости, но значительно усилит взлом, сведя его практически на нет (согласно частному случаю эмпирического закона Парето, в течение 80% времени работы процессор выполняет 20% от общего числа реализованных в программе команд).

Для еще большего усложнения анализа кода можно применить так называемый «метод перемешивания» кода. Пусть команда виртуальной машины состоит из кода команды и указателя на следующую обрабатываемую команду (при этом сам код команды может быть зашифрован, используя в качестве ключа текущий и/или следующий указатель):

```
struct command
{
    CODE code;
      struct command* next_command;
}
```

Таким образом, память виртуальной машины будет представлять собой связанный список, что позволит: а) легко перемешивать исполняемый код; б) эффективно добавлять и удалять не значащие (т.н. "мусорные") команды. В итоге получаем чрезвычайно простой, но весьма нетривиальный полиморфный код, для анализа которого взломщикам придется написать специальный обработчик. Можно также реализовать память виртуальной машины в виде двоичного дерева, аргументы передавать через списки, выполнение инструкций организовать в виде дека.

Рассмотрим наиболее устойчивый вариант защиты — виртуальные машины с произвольно генерируемым набором инструкций. Здесь любую устойчивую комбинацию команд можно заменить одной составной инструкцией (это также минимизирует объем исполняемого кода). Если пойти дальше, то

можно наугад взять несколько инструкций, даже не обязательно соседних, и заменить их одной новой, продолжая так сколь угодно долго. Во втором поколении инструкций окажется большой выбор инструкций для дробления. При условии, что такое дробление будет осуществляться по случайному закону, полученные инструкции, вероятнее всего, не совпадут с исходными, в результате чего второе поколение инструкций ВМ станет практически неузнаваемо [7].

При этом проанализировать логику такой виртуальной машины невероятно трудно. Дело в том, что логический смыл команд, сгенерированных произвольным образом в подавляющем большинстве случаев, не ясен, т. к. число связей между отдельными командами стремится к n!, где n— количество исходных команд. Поскольку виртуальная машина может оперировать сотнями и даже тысячами команд, ни проанализировать, ни дизассемблировать код невозможно – все сводится к нестандартности ВМ, алгоритму ее поведения [7].

Основываясь на сделанных выше предположениях и утверждениях, подведем промежуточные итоги, и представим обобщенную схему защиты приложения (рис. 3).

Рис. 3. Обобщенная схема защиты приложения

Выводы

Разработан принципиально новый подход к построению систем защиты исполняемого программного кода. Предложен метод защиты программного обеспечения на базе виртуальной машины. Показана высокая устойчивость такой подсистемы защиты к взлому.

Потенциал технологии виртуальных машин в области защиты информации очень высок. Внедрять сложную аппаратную защиту не всегда целесообразно исходя из затрат времени, средств и сложности внедрения. Программная же реализация обладает чрезвычайной гибкостью и простотой, ограничиваясь лишь средствами математики, языков программирования и, конечно, профессионализмом разработчиков.

Список литературы

1. Козлов Д. Технология криптозащиты кода [Электронный ресурс] / Дмитрий Козлов. – Режим доступа к документу: http://hardline.ru.

2. Список популярных дизассемблеров [Электронный ресурс]. – Режим доступа к документу: http://wasm.ru/toollist.php?list=13.

3. UPX: the Ultimate Packer for eXecutables Электронный ресурс]. – Режим доступа к документу: http://upx.sourcefourge.net.

4. Электронный ресурс]. – Режим доступа к документу: http://www.aspack.com/asprotect.aspx.

5. Электронный ресурс]. – Режим доступа к документу: http://ru.wikipedia.org/wiki/Виртуальная_машина.

6. Электронный ресурс]. – Режим доступа к документу: http://ru.wikipedia.org/wiki/Шитый_код.

7. Петров А.С. Технология защиты программного кода посредством применения виртуальной машины / А.С. Петров, А.А. Петров // Вестник ВНУ. – 2009. – № 9 (103), часть 1. – С. 117-122.

Поступила в редколлегию 19.03.2010

Рецензент: д-р техн. наук, проф. В.А. Лужецкий, Винницкий национальный технический университет, Винница.

МЕТОДИ ЗАХИСТУ ПРОГРАМНОГО КОДУ

О.С. Петров, А.О. Петров

Розглянуто питання побудови системи захисту програмного коду. Описано методики порушення цілісності виконуваного програмного коду і способи їх усунення. Розглянуто застосування технології віртуальних машин в контексті захисту виконуваного коду від аналізу. Дано обґрунтування застосування віртуальної машини як оптимального способу протидії аналізу виконуваного коду.

Ключові слова: *програмний код, захист виконуваного коду, віртуальна машина, інтерпретація, дизасемблювання.*

METHODS OF DEFENCE OF PROGRAMMATIC CODE

A.S. Petrov, A.A. Petrov

The questions of construction of the system of defence of programmatic code are considered. The methods of violation of integrity of executable programmatic code and methods of their removal are described. Application of technology of virtual machines is considered in the context of defence of executable code от an analysis. The ground of application of virtual machine is given as an optimum method of counteraction the analysis of executable code.

Keywords: *programmatic code, defence of executable code, virtual machine, interpretation, disassembling.*