А. А. Краснопевцев
Московский инженерно-физический институт (государственный университет)

РАЗРАБОТКА АВТОМАТИЧЕСКОЙ ЗАЩИТЫ ОТ НЕСАНКЦИОНИРОВАННОГО КОПИРОВАНИЯ .NET ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ВНЕШНЕГО АППАРАТНОГО МОДУЛЯ

Целью данной работы являлось рассмотрение способа защиты .NET приложений от копирования при помощи внешнего аппаратного модуля. В рамках статьи было проведено описание данного способа, рассмотрены его положительные и отрицательные стороны. В статье рассматривается программный комплекс, реализующий защиту .NET приложений от копирования разработанным способом.

Защита программного обеспечения до момента появления систем разработки приложений типа технологии .NET фирмы Microsoft или технологии Java фирмы Sun Microsystems, зачастую сводилась к использованию недостатков существующей и применяемой на данный момент архитектуры представления кода и данных для процессора, а именно: архитектуры Фон-Неймана [1, 2]. Основной недостаток этой архитектуры представления данных и инструкций приложения заключается в том, что не существует реальной возможности отличить код приложения от используемых им данных.

Однако при появлении таких сред разработки приложений, как, например, .NET, использовать старые подходы для защиты программного обеспечения от несанкционированного копирования стало нецелесообразным [3, 4]. Это связано с тем, что .NET приложения, так называемые сборки, в отличие от «обычных», т. е. компилируемых в машинный код приложений, имеют достаточно четкую структуру хранения и преставления кода и данных. Вследствие этого отличить код приложения от обрабатываемых этим приложением данных в случае с .NET сборками не представляется сложным. Кроме того, вследствие удобства и простоты разработки приложений с применением технологий типа .NET эти технологии пользуются достаточно большой популярностью среди разработчиков программного обеспечения и с их использованием создается достаточно большое количество приложений.

В данной статье предложен метод защиты приложений, компилируемых в байт-код, при помощи вынесения части кода приложения во внешний аппаратный модуль, недоступный для чтения и модификации. Помимо этого, в рамках данной статьи рассматривается программная реализация вышеуказанного метода для защиты .NET приложений от копирования.

При рассмотрении различных методов, используемых для защиты приложений от копирования, было выявлено, что все эти методы в той или иной мере базируются на подходах

«черного» или «серого» ящиков [4]. Этот тезис можно обосновать с той точки зрения, что любая существующая система защиты приложений от копирования построена таким образом, чтобы скрыть от злоумышленника либо «запутать» код приложения или какие-либо части кода этого приложения.

На основании данного вывода был разработан метод защиты приложений, компилируемых в байт-код, т. е. некоторое промежуточное представление, при помощи вынесения части кода во внешний аппаратный модуль [5]. В разработанном методе защиты в виде аппаратного модуля выступает электронный ключ с загружаемым кодом. В отличие от других электронных ключей, электронные ключи с загружаемым кодом имеют одну достаточно интересную особенность. Эта особенность заключается в том, что кроме предоставления стандартных функций «обычного» электронного ключа, таких, например, как аппаратно реализованные алгоритмы шифрования, электронный ключ с загружаемым кодом способен выполнять некоторый загруженный в него код. Причем реализация электронного ключа с загружаемым кодом такова, что код, который находится внутри этого ключа, не может быть с него считан и каким-либо образом модифицирован. Так, рассматриваемый электронный ключ является наиболее близкой к идеальной реализацией «черного ящика».

Разработанный метод защиты основан на том, что часть кода .NET сборки выносится в электронный ключ с загружаемым кодом. Таким образом, для компрометации защищенного разработанным способом приложения злоумышленнику придется анализировать алгоритм, заложенный в электронном ключе, используя методы атак на классический «черный ящик», а в дальнейшем восстанавливать алгоритм. Простое «отсоединение» приложения от электронного ключа, например, при помощи модификации драйвера, в случае с электронным ключом с загружаемым кодом не приведет к компрометации приложения. Это связано не только с тем, что защищенное приложение использует электронный ключ на предмет проверки своей лицензии, но также и с тем, что он необходим для полноценной работы приложения.

Среди положительных сторон разработанного метода защиты приложений от копирования важно выделить то, что такой метод защиты «ориентирован» на приложения, компилируемые в байт-код. Это связано с тем, что для защиты приложения код, который подлежит выгрузке на электронный ключ, нужно преобразовать к ассемблерным инструкциям электронного ключа. Таким образом, необходимым шагом при защите будет являться декомпиляция кода приложения и компиляция этого кода в инструкции ассемблера ключа. Как известно, вследствие использования архитектуры Фон-Неймана декомпиляция приложения, компилируемого в машинный код, в общем случае невозможна. Однако в случае с приложениями, компилируемыми в некоторое промежуточное представление, декомпиляция кода осуществима, поэтому можно защитить приложения от копирования рассматриваемым в данной статье способом.

Кроме «ориентированности» разработанного метода на компилируемые в байт-код приложения, достаточно важным положительным аспектом является сложность компрометации защищенного приложения.

Однако, помимо положительных сторон, у данного метода защиты программного обеспечения от копирования существуют и отрицательные стороны. К минусам данного способа можно отнести общее уменьшение скорости работы приложения, которое связано с тем, что защищенное приложение во время работы периодически обращается к электронному ключу с загружаемым кодом и при этих обращениях часть кода выполняется на процессоре электронного ключа. На момент написания статьи электронные ключи с загружаемым кодом оснащались процессорами архитектуры ARM, имеющими достаточно низкую производительность, что, в свою очередь, влияло на общую скорость выполнения защищенного приложения.

Среди отрицательных сторон разработанного метода можно также выделить то, что не все функции приложения могут быть вынесены во внешний аппаратный модуль. Основных причин

для возникновения такого ограничения две. Первая и основная причина состоит в том, что функция, которая подлежит перенесению на электронный ключ с загружаемым кодом, не должна иметь внешних вызовов, т. е., например, обращаться к каким-либо внешним классам. Данное ограничение связано с тем, что при попытке обращения к какому-либо внешнему, относительно приложения в электронном ключе, объекту данный объект не будет найден и, следовательно, приложение не сможет корректно функционировать. Таким образом, в общем случае для вынесения во внешний аппаратный модуль больше всего подходят чисто алгоритмические функции, т. е. функции, которые реализуют какой-либо математический алгоритм. Для устранения данного недостатка можно использовать разработанный способ защиты совместно с «классическими» способами. Думается, что наиболее подходящим в данном случае будет использование виртуальной машины и псевдокода.

Вторая причина заключается в том, что небезопасно выносить в электронный ключ с загружаемым кодом тривиальные алгоритмы, т. е. алгоритмы, которые будут элементарно восстанавливаться при анализе входных и выходных данных. В случае если поместить во внешний аппаратный модуль именно такой, тривиальный, алгоритм, то злоумышленник сможет его восстановить и в дальнейшем создать полный эмулятор ключа и скомпрометировать приложение в целом. Кроме того, желательно выносить в электронный ключ именно алгоритмы, а не таблицы с запросами и ответами, что может быть чревато построением табличного эмулятора, который в данном случае будет являться полным эмулятором.

Однако при некоторых недостатках, возникающих при использовании данного метода защиты, он имеет достаточно высокую степень надежности защиты приложения от копирования. Такая надежность связана с тем, что, как говорилось выше, электронный ключ с загружаемым кодом является наиболее близкой физической реализацией идеального «черного ящика». Таким образом, для компрометации приложения злоумышленнику придется анализировать алгоритм, который находится внутри внешнего аппаратного модуля, только путем анализа зависимости входных и выходных значений. Вследствие этого разработанный метод является достаточно сложно компрометируемым.

На основе разработанного метода был создан программный комплекс, позволяющий осуществлять защиту .NET приложений от несанкционированного копирования при помощи вынесения части кода приложения во внешний аппаратный модуль, т. е. электронный ключ с загружаемым кодом [6]. Комплекс состоит из нескольких основных компонентов, каждый из которых отвечает за выполнение своего класса функций. Всего в этой системе защиты от копирования пять основных компонентов, а именно:

- дизассемблер;
- анализатор классов и процедур приложения;
- декомпилятор;
- компилятор в ARM инструкции;
- утилита работы с электронным ключом.

Одним из наиболее важных модулей в разработанном комплексе является дизассемблер, основной его функционал сводится к преобразованию метаданных .NET приложения к представлению, удобному для дальнейшей работы системы защиты. Таким образом, во время работы на вход дизассемблер получает приложение, а на выходе генерируется внутреннее представление всех метаданных сборки. После дизассемблирования полученное представление передается на вход анализатору классов и методов.

Основной функционал анализатора классов и процедур приложения заключается в анализе представленных ему данных и построении некоторой иерархии метаданных, т. е. выявлении между ними взаимосвязей, определении вызовов внутри приложения и отделении внешних вызовов. Кроме

того, данный модуль способен отличать функции, которые могут быть вынесены в электронный ключ с загружаемым кодом, от функций, которые не могут быть вынесены вследствие описанных ранее ограничений разработанного метода. После того, как будет проведен анализ метаданных приложения, установлены все взаимосвязи метаданных между собой, полученная иерархическая архитектура представляется в виде дерева пользователю, с тем чтобы пользователь мог указать, какие из функций, «одобренных» анализатором для защиты, подлежат защите, а какие нет. После того, как пользователь выберет необходимые функции, оригинальный код из данных функций удаляется и на его место ставятся «заглушки», основная задача которых сводится к тому, чтобы передавать входные параметры метода в код, который будет выполняться в электронном ключе, и возвращать результаты работы данной функции обратно в приложение. После этого вынесенный из оригинальной функции код передается декомпилятору, который преобразует IL код функции в аналогичный код на языке высокого уровня (в данной реализации C++).

При помощи данного модуля полученный код будет восстановлен алгоритмически идентично, т. е. логика работы приложения не поменяется, однако при этом могут быть изменены имена локальных переменных, так как их оригинальные имена в скомпилированном .NET приложении не хранятся.

Полученный от предыдущего модуля код на языке C++ затем передается компилятору в ARM инструкции, основной функционал компилятора заключается в том, чтобы из кода на каком-либо языке получать файлы, исполняемые в электронном ключе.

После компиляции полученный файл при помощи утилиты работы с электронным ключом переносится во внешний аппаратный модуль.

Алгоритм защиты приложения выглядит следующим образом: защищаемое приложение дизассемблируется, т. е. преобразуется в некоторое внутреннее представление комплекса защиты, затем выделяются взаимосвязи метаданных этого приложения и определяются функции, которые могут быть защищены с использованием такого подхода. После этого из функций, подлежащих защите, удаляется оригинальный код, и на его место устанавливаются заглушки, обеспечивающие работоспособность защищенного приложения. Затем вынесенный код декомпилируется, полученный в результате код компилируется в набор инструкций ARM процессора, а потом полученный файл переносится на электронный ключ с загружаемым кодом.

Алгоритм работы защищенного приложения выглядит так: сначала параметры, переданные защищенной функции, собираются в массив, затем производится поиск электронного ключа с загружаемым кодом, потом происходит передача массива с параметрами функции, после чего обратно в приложение передается результат работы защищенной функции.

Таким образом, после защиты получается приложение, по функционалу идентичное незащищенному, часть функционала которого, однако, вынесена на электронный ключ с загружаемым кодом.

В рамках данной статьи был рассмотрен метод защиты приложений, компилируемых в промежуточное представление, от копирования. Были рассмотрены положительные и отрицательные стороны данного метода, а также ограничения, которые разработанный способ защиты налагает на функции, подлежащие защите.

Кроме того, было приведено описание разработанного программного комплекса, автоматизирующего способ защиты приложений от копирования путем вынесения части кода приложения во внешний аппаратный модуль. Были рассмотрены алгоритм защиты приложения и алгоритм работы уже защищенного разработанным методом приложения.

СПИСОК ЛИТЕРАТУРЫ:

1. *Ахо А., Ульман Дж.* Компиляторы: принципы, технологии и инструменты. М.: Издательский дом «Вильямс», 2003. — 768 с.
2. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. М.: Мир, 1978.
3. *Холанд Г., Мак-Гроу Г.* Взлом программного обеспечения анализ и использование кода. М.: Издательский дом «Вильямс», 2005. — 400 с.: ил.
4. *Матросов А. А.* Курс лекций «Защита программного обеспечения». URL: http://aktivco.ru/course/lecture/.
5. *Краснопевцев А. А.* Разработка средств автоматизации для переноса байт-кода во внешний аппаратный модуль // Технологии Microsoft в теории и практике. М.: Вузовская книга, 2008. С. 139—141.
6. *Краснопевцев А. А., Букасов В. А.* Разработка средств автоматической защиты приложений, содержащих байт-код // Материалы 10-й Международной научно-практической конференции, Таганрог, 24—27 июня 2008. С. 15—16.

А. А. Станкевичус
Московский инженерно-физический институт (государственный университет)

ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО ПРОВЕРКИ БЕЗОПАСНОСТИ КОДА, ПРЕДНАЗНАЧЕННОГО ДЛЯ ИСПОЛНЕНИЯ В ГРИД-СЕТЯХ

В работе анализируются методы и средства защиты Грид-сетей от вредоносного кода, их преимущества и недостатки. Предлагается новое инструментальное средство, позволяющее в автоматическом режиме подтверждать безопасность программ на языке Си, предназначенных для Грид-сетей.

Введение

При работе с Грид-сетями существует угроза распространения вредоносного кода производителем прикладного ПО Грид. Вредоносный код может нарушать работу отдельных ресурсов Грид, а также Грид в целом, что определяет актуальность разработки средств защиты Грид от вредоносного кода. Задачей настоящей работы является разработка инструментального средства проверки безопасности кода, предназначенного для Грид-сетей.

1. Инструментальные средства защиты Грид-сетей от вредоносного кода

Проверка безопасности кода для Грид-сетей нередко производится вручную, однако многие специалисты считают целесообразным применение инструментальных средств.

Для защиты Грид-сетей от вредоносного кода могут быть применены антивирусные программы, такие как [1], но современные антивирусные средства во многих случаях не выявляют новые виды вирусов, а также вредоносное ПО, не являющееся вирусом или сетевым червем.

Для защиты Грид от вредоносного кода могут быть использованы верификаторы, например [2, 3]. Однако для успешного применения многих средств формальной верификации необходима формализация верифицируемых свойств для конкретной программы, что само по себе может представлять сложную задачу, требующую участия человека, заинтересованного в обеспечении безопасности узла Грид.

Для защиты Грид от вредоносного кода могут быть применены безопасные языки, такие как [4], и инструментальные средства на их основе. К сожалению, производительность программ на безопасных языках часто оказывается ниже производительности аналогичных программ на небезопасных языках, таких как язык Си. Применение разработчиками ПО безопасных языков требует их предварительного изучения. Перевод существующего кода на безопасный язык может быть длительной и дорогостоящей процедурой.

Для защиты Грид от вредоносного кода могут быть применены виртуальные машины [5, 6]. При этом производительность прикладного ПО, выполняемого на виртуальной машине, снижается. Кроме того, безопасность многих виртуальных машин может быть поставлена под сомнение [7].