УДК 004.658.2; 004.056.53

Кокоулин А.Н., Салимзебаров Э.Д.

МЕТОДЫ ЗАЩИТЫ ОТ ИЗМЕНЕНИЙ И ДИЗАССЕМБЛИРОВАНИЯ .NET ПРИЛОЖЕНИЙ

Пермский национальный исследовательский политехнический университет, Пермь, Поздеева 7, 614013

Kokoulin A. N., Salimzebarov E. D.

METHODS FOR .NET APPLICATIONS PROTECTION AND DISASSEMBLING

Perm National Research Polytechnic University, Perm, Pozdeeva 7, 614013

Аннотация. В работе рассматривается декомпиляция .Net приложений и различные способы защиты, такие как обфускация, упаковщики, контроль запуска в отладке и применение электронных ключей для защиты контроля запуска. Так же оценивается степень надежности и устойчивости ко взлому рассмотренных средств защиты.

Ключевые слова: .Net приложения, обфускация, способы защиты

Abstract. The paper considers .Net applications decompilation process and different methods of protection against decompilation such as obfuscation, packers, launch control and electronic keys. Also we estimate the level of reliability of these protection approaches.

Key words: .Net application, obfuscation, methods of protection.

Введение

На сегодняшний день популярность и усовершенствование .Net платформы обуславливает ее широкое распространение в настольных и серверных Windows-системах. В связи с активной политикой Microsoft по развитию и внедрению платформы .Net, все больше разработчиков ведут разработку приложений для этой платформы, и в новых версиях Windows runtime-

модули .Net Framework актуальных версий уже предустановлены в операционной системе. В связи с этим проблема защиты приложений от воздействия злоумышленников актуальна.

Декомпиляция и дизассемблирование

Рассмотрим независимо друг от друга задачу дизассемблирования и задачу декомпиляции программ. Под декомпиляцией понимается построение программы на языке высокого уровня, эквивалентной исходной программе на языке низкого уровня (языке ассемблера). Под дизассемблированием понимается построение программы на языке ассемблера, эквивалентной исходной программе в машинном коде. Программа в машинном коде представляется либо в виде исполняемого модуля в стандартном для целевой операционной системы, либо в виде дампа содержимого памяти, либо в виде трассы исполнения программы.

При дизассемблировании выполняется трансляция исполняемого файла, представляемого в виде набора машинных команд, в программу на языке ассемблера. При декомпиляции программа с представления низкого уровня транслируется в представление высокого уровня. Дальнейшим этапом повышения уровня абстракции программы может быть рефакторинг.

Чаще всего дизассемблер используют для анализа программы (или ее части), исходный текст которой неизвестен — с целью модификации, копирования или взлома. Реже — для поиска ошибок (багов) в программах и компиляторах, а также для анализа и оптимизации создаваемого компилятором машинного кода.

Стоит отметить, что .Net приложения компилируются на язык CIL который, является настоящим "родным" языком для платформы .NET. При создании .NET-сборки с помощью того или иного языка (C#, VB, COBOL.NET и т.д.) соответствующий компилятор всегда преобразует исходный код на этом языке в код на CIL. Так же и декомпиляторы уже работают с языком CIL, что позволяет восстанавливать исходный код программы чрезвычайно точно.

Рассмотрим рисунок 1, где представлен исходный код и код открытый

декомпилятором .Net. По сравнению с реверсом и отладкой Win32-приложений, задача декомпиляции .Net доступна даже малоквалифицированному специалисту.

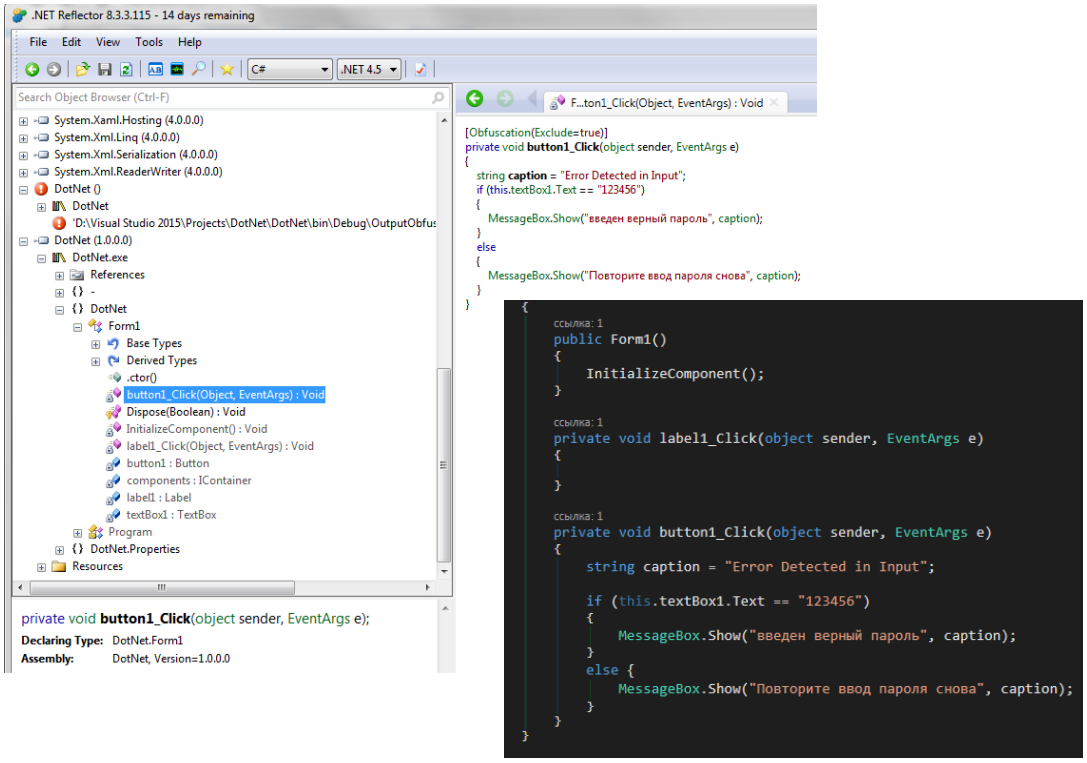

Рис. 1. Пример кода программы полученного декомпилятором.

Защита от модификаций

Пожалуй, самый распространенный метод защиты — это применение обфускаторов[2]. Обфускатор - приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

Некоторые обфускаторы предоставляют функциональность, которая способна защитить приложение от модификаций. Такой механизм часто называют tamper protection или tamper defense. В этом случае программа будет корректно выполнятся только в том случае если они не была изменена злоумышленником. Если какие-то изменения были – программа просто перестанет работать, либо будет выводить некоторое сообщение о недопустимости использования модифицированной программы.

Хотя подобный метод помогает поможет сделать иллюзию защиты, но это не значит, что вскрыть скомпилированное приложение будет сложно. На данный момент существует много различных так называемых «деобфускаторов» пример такого «CodeWall» которые вы пару кликов мыши восстановят исходный код программы. Выходом может служить только ручная настройка со своими методами, которых хакерские программы воспринимать не будут.

Защита от дизассемблирования

Несмотря на то, что в процессе обфускации структуры исполняемого кода программы претерпела серьезные изменения, полученный код в результате работы дизассемблирования[1], будет чрезвычайно сложен и запутан, что потребует долгие часы разбора. Так как злоумышленник ищет легкие пути обхода, он наверняка не думает сразу запускать дизассемблер и пытаться что- то разобрать, а воспользовавшись уже готовым ПО таки как «.Net Reflector» [3] получит более читаемый код и иногда и полностью восстановлен, но перед этим обязательно придется деобфусцировать приложение. Пример работы обфускатора представлен на рисунке 2.

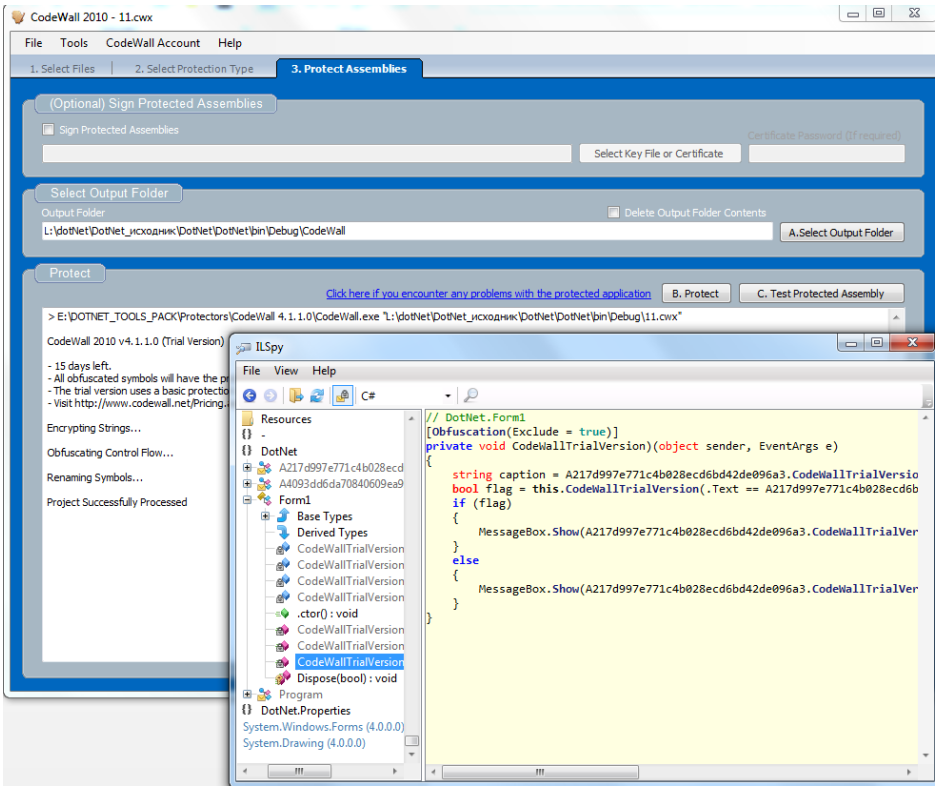

Рис. 2. Пример работы программы деобфускации.

Исходя из такой ситуации можно посоветовать применять различные обертки и упаковщики. Примером обертки для .Net приложений может служить «win32» она не позволит воспользоваться специальным ПО для взлома .Net приложений, что заставит злоумышленника воспользоваться не удобным для него дизассемблером.

В дополнении к тому что дизассемблер будет видеть код в машинных кодах, но и к тому же он еще способен запустить приложение под отладкой, а вот это уже очень серьёзная угроза безопасности. Решением данной проблемы есть установка так называемых «breakload» это простой вариант вставки в тело каждого метода дополнительного фрагмента кода, который содержит некоторую некорректную или несуществующую инструкцию. При загрузке метода, у которого добавлен такой фрагмент, дизассемблер не сможет распознать эту инструкцию и соответственно не сможет загрузить тело метода для анализа злоумышленником. А для того, чтобы данная инструкция не влияла на процесс выполнения программы ее следует оградить инструкциями безусловных переходов таким образом, чтобы она никогда не смогла получить управление. Такой метод безусловно осложнит жизнь злоумышленники и заставит его копаться в компилированных кодах

Более продвинутый способ может выглядеть следующим образом. Код метода шифруется с помощью некоторого алгоритма шифрования и результат шифрации помещается либо в ресурсы, либо вовсе в отдельный файл. Исходное тело метода замещается кодом некоторого загрузчика, который знает где находится зашифрованный фрагмент кода и способ как его расшифровать. При вызове такого метода загрузчик получает управление, он дешифрует реальный код, и выполняет его. Таким образом при дизассемблировании исследовать этот код будет абсолютно невозможно.

Так же стоит отметить что защитить программный продукт можно с помощью HASP ключей, которые будут намного надежнее всяких программных методов. Но это не означает что их так же невозможно обойти. Для обхода HASP ключей используются различные эмуляторы подключения ключа. Это

намного трудозатратно и требует высокой квалификации. Примером программы для защиты с помощью HASP может служить HASP SRM Vendor Suite (см рисунок 3).

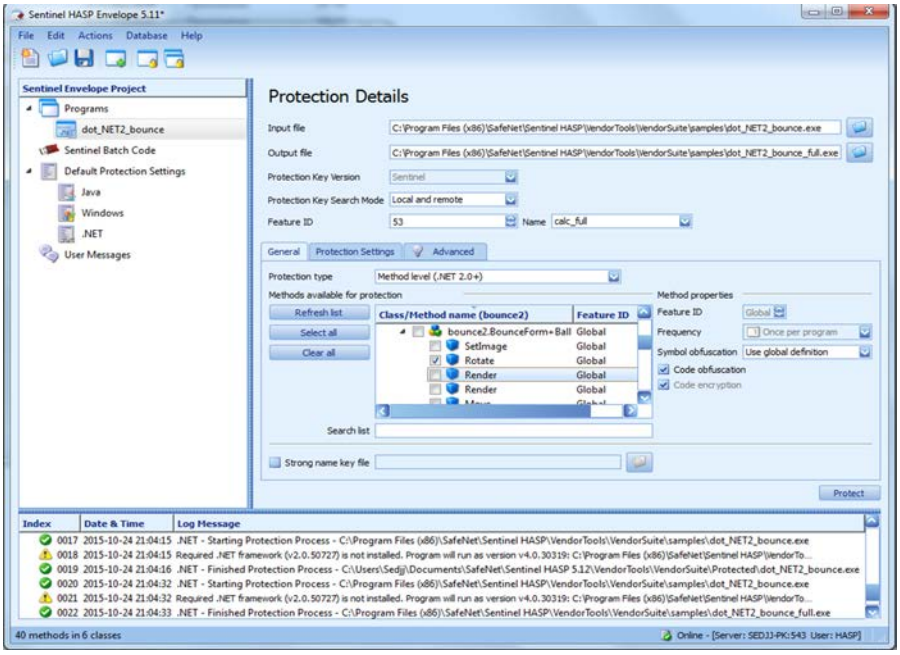

Рис. 3. Пример программы для защиты с HASP

Заключение

В данной статье были рассмотрены различные методы взлома и некоторые методы для защиты .Net приложений. Подводя итоги можно сказать что на данный момент вариантов взлома довольно много, что существенно усложняет жизнь разработчиков .Net приложений. Им приходится придумывать различные методы защиты своего ПО, иначе затраченные деньги уходят в некуда. Но стоит так же отметить что существует довольно серьезные методы защиты ПО, что не дает отчаиваться разработчикам и заплатив за лицензию небольшие деньги или дописать самому модуль защиты, из исходников, которые выложены в Open source можно не переживать что любой желающий сможет взломать программу. Подводя итоги можно сказать что разработка под .Net платформу приносит много проблем при попытке заработать денег на своем ПО и поэтому рекомендуется несколько раз подумать прежде чем приступать.

Литература:

1. SecurityLab: [Электронный ресурс]: Реверс-инжиниринг NET-приложений. Часть первая: Введение – URL: http://www.securitylab.ru/analytics/439107.php (дата обращения 24.11.15).

2. Netobf: [Электронный ресурс]: Как защитить приложение на .NET (C#, VB.NET) – URL: http://netobf.com/how-to-protect-the-application (дата обращения 24.11.15).

3. SecurityLab: [Электронный ресурс]: Обратная инженерия с помощью Reflector. Часть 1– URL: http://www.securitylab.ru/analytics/451094.php (дата обращения 24.11.15).

4. Kaimi. блог: [Электронный ресурс]: Патчинг софта, накрытого .NET Reactor'ом, на примере ActualSpamPro – URL: http://kaimi.ru/2010/11/net-reactor-actualspampro/ (дата обращения 24.11.15).

5. Scientific World: [Электронный ресурс]: Безопасность баз данных oracle в многозвенных информационных системах– URL: http://www.sworld.com.ua/index.php/ru/technical-sciences-m115/informatics-computer-science-and-automation-m115/25323-m115-237 (дата обращения 25.11.15).

6. Даденков С.А., Кон Е.Л. Оценка степени влияния некоторых факторов на производительность LonWorks сети // Образовательные ресурсы и технологии. 2014. № 2 (5). С. 72-76

7. СибАК: [Электронный ресурс]: Обзор сетевого протокола teredo на основе стандарта rfc 4380 – URL: http://sibac.info/19640 (дата обращения 24.11.15).

Дата отправки: 26.11.2015

Научный руководитель: к.т.н., Кокоулин А. Н.