УДК 681.3.06, 681.324.06

ЛЕКСИЧЕСКАЯ ОБФУСКАЦИЯ КАК ИНСТРУМЕНТ ДЛЯ ВНЕДРЕНИЯ ВОДЯНЫХ ЗНАКОВ

Ю.Г. ВАШИНКО[1], А.В. БУДНИК[2], В.М. БОНДАРИК[3]

[1]ООО «СКБ «Радиотехпроект»,
ул. Огинского, 6–36, Минск, 220114, Беларусь

[2]Учреждение образования «Белорусская государственная академия связи»,
ул. Ф. Скорины, 8/2, Минск, 220114, Беларусь

[3]УО «Белорусский государственный университет информатики и радиоэлектроники»,
ул. П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 22 мая 2019

Рассмотрены понятия обфускации и водяных знаков. Проведено исследование программ, которые позволяют осуществлять обфускацию и внедрение водяных знаков одновременно. Предложены подходы к использованию лексической обфускации в качестве инструмента для внедрения водяных знаков.

Ключевые слова: обфускация, водяной знак, защищенное кодирование.

Введение

Проблема защиты авторского права на программный продукт является одной из наиболее актуальных, решение которой занимает значительную часть расходов компаний-производителей программного обеспечения (ПО). Основными источниками нелицензионного ПО являются сети с анонимным доступом, а также торрент-сети (torrent-network), которые получили широкое распространение в сети Интернет [1, 2]. Любой пользователь глобальной сети может получить доступ к такого рода сетям, где можно отыскать большинство существующих программных продуктов с приложенными к ним утилитами для взлома.

Как правило, производители выбирают традиционные способы защиты, такие как использование серийных номеров и шифрование программы некоторым криптографическим алгоритмом с дешифрованием ее «на лету» при запуске. Однако данные методы имеют свои недостатки. Так, серийные номера могут распространяться среди незаконных пользователей программы, а дешифрованный код программы может быть считан прямо из памяти исполняющего устройства. Другой возможностью для злоумышленников является обратное проектирование программных средств, которое позволяет найти и исключить из программы участки, определяющие верность введенного пользователем серийного номера, либо обнаружить криптографические ключи, которые используются для дешифрования программы. Под понятием обратного проектирования (reverse engineering) подразумевается процесс выяснения технологических принципов устройства, объекта или системы при помощи анализа его структуры и функционирования [3, 4].

Особенно актуальна проблема обратного проектирования для приложений технологий .NET, где исполняемые файлы представляют собой сборки с метаданными, написанные на промежуточном языке (intermediate language) [5]. Данные сборки хранят в себе достаточно информации для восстановления исходного кода программы с сохранением имен классов, методов и их параметров.

Основным способом защиты ПО от обратного проектирования является обфускация, главная цель которой заключается в том, чтобы запутать программный код и устранить

большинство логических связей в нем, трансформировать его так, чтобы он был труден для изучения и модификации посторонними пользователями [6].

Одно из наиболее точных и математически описанных определений обфускации дано в работе [7].

Пусть P – множество всех программ. Обфускацией будем называть функцию $O: P \rightarrow P'$, для которой соблюдаются следующие условия:

1) Для любой исходной программы $p \in P$, $p' = O(p)$, если p не может завершить работу в силу возникшей ошибки, то p' при возникновении идентичной ошибки может прекратить или не прекращать работу. Во всех остальных случаях p' должна завершать работу и выдавать результат эквивалентный результату программы p;

2) Время выполнения p' должно быть полиномиально сравнимо со временем выполнения p;

3) Время восстановления исходной программы p из p' должно как минимум превышать время разработки программы p.

Одним из самых простых видов обфускации является лексическая обфускация, суть которой заключается в форматировании кода программы, изменении его структуры таким образом, чтобы он стал нечитабельным, менее информативным и трудным для изучения [6]. Базовым способом лексической обфускации является переименование имен классов, функций, полей, параметров, переменных.

Обфускация позволяет разработчикам повысить уровень защиты своего ПО от обратного проектирования и тем самым защитить свою интеллектуальную собственность. Однако в последние несколько лет на ряду с обфускацией происходят попытки применения технологии водяного знака (ВЗ) для защиты прав интеллектуальной собственности на программное обеспечение, что уже широко используется в индустрии производства мультимедийного контента (видео, аудио, изображения).

Применительно к внедрению в ПО водяной знак – это сообщение, состоящее из набора бит (0 и 1), имеющее конечную длину $length \geq 0$ [7].

Процесс внедрения ВЗ в работе [7] сформулирован таким образом: пусть P – множество программ, а W – множество водяных знаков. Тогда функция $A: P \times W \rightarrow P$ называется функцией (алгоритмом) внедрения. Если $p' = A(p, w)$, где $p \in P$ и $w \in W$, то p' – программа с внедренным водяным знаком.

Комбинирование внедрения водяных знаков и обфускации

С учетом того, что рынок программного обеспечения, использующего промежуточные языки, растет быстрыми темпами, все более востребованными становятся программы и утилиты для защиты ПО от несанкционированного использования. В большинстве своем для защиты используется обфускация или внедрение ВЗ, однако на рынке программного обеспечения существует ряд продуктов, которые для повышения уровня защиты от несанкционированного использования позволяют применять ВЗ и обфускацию кода одновременно: *Spices.Net Obfuscator* и *Crypto Obfuscator For .Net.*

Несмотря на то, что основные функциональные возможности программы *Spices.Net Obfuscator* направлены на обфускацию, данная программа также позволяет внедрять один водяной знак для каждого обфусцируемого модуля. Для исследования стойкости был проведен анализ обфусцированного исполняемого файла на основе его текстовой версии, полученной при помощи утилиты *IL Disassembler*, входящей в пакет установки *Visual Studio*. Исходная строка ВЗ записана в 16-ричном формате кодировки UTF-8 (рис. 1), что позволяет сделать заключение о том, что применяемый метод внедрения ВЗ является недостаточно устойчивым, а сам ВЗ может быть подвержен модификации.

```
                76 61 6C 75 61 74 69 6F 6E 20 76 65 72 73 69 6F   // valuation versi
                6E 29 20 61 6E 64 20 63 61 6E 27 74 20 62 65 20   // n) and can't be
                75 73 65 64 20 66 6F 72 20 63 6F 6D 6D 65 72 63   // used for commer
                69 61 6C 20 70 75 72 70 6F 73 65 73 2E )          // ial purposes.
.
014 */ = ( 01 00 0E 63 6F 6D 70 61 6E 79 20 73 74 72 69 6E   // ...company strin
           67 )                                               // g
tribute/*0100001B*/::.ctor(string) /* 0A00000D */ = ( 01 00 07 31 2E 30 2E 30 2E 30
```

Рис. 1. Водяной знак в файле промежуточного языка

Основным предназначением программы *Crypto Obfuscator For .Net* также является обфускация, однако функционал внедрения ВЗ позволяет использовать до 10 ВЗ для одного модуля. На рис. 2 показано диалоговое окно, которое позволяет вводить водяные знаки.

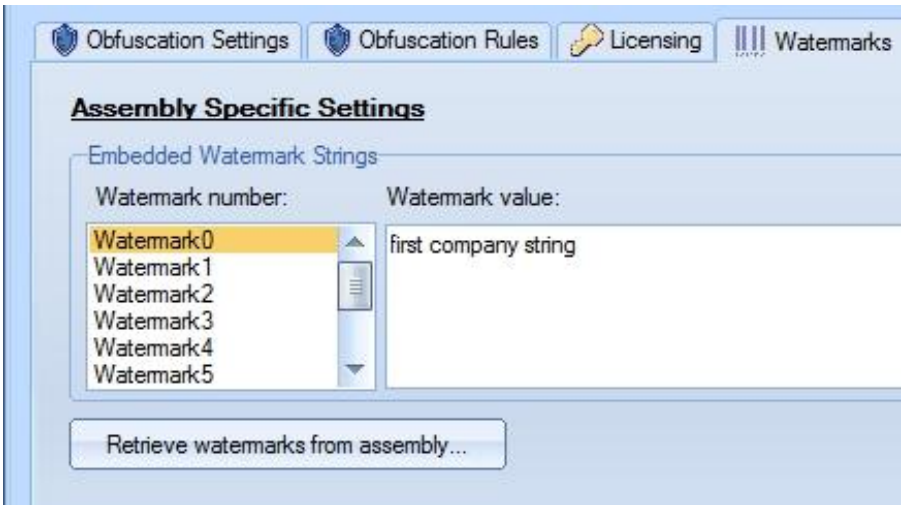

Рис. 2. Настройка водяных знаков в Crypto Obfuscator For .Net

Исследование полученного исполняемого файла при помощи утилиты *IL Disassembler* позволило обнаружить внедренные ВЗ в открытом виде (рис. 3), что говорит о слабой защищенности ВЗ знаков от удаления и модификации.

```
// User Strings
// -------------------------------------------------------
// 70000001 : (135) L"."3D9B94A98B-76A8-4810-B1A0-4BE7C4F9C98DA2#.first company
string!@#$%second company string!@#$%!@#$%!@#$%!@#$%!@#$%!@#$%!@#$%!@#$
// %!@#$%"
//              User string has unprintables, hex format below:
//              0011 0022 0033 0044 0039 0042 0039 0034 0041 0039 0038 0042 002d 0037 0036
0041
```

Рис. 3 Водяные знаки в файле промежуточного языка

Однако наиболее востребованными методами внедрения ВЗ являются алгоритмы, которые не позволяют обнаружить ВЗ при просмотре дизассемблированного кода и требуют дополнительного изучения. Наиболее подробно такого рода алгоритмы описаны в работах [7-10].

В рамках данной работы предлагается в качестве метода внедрения ВЗ рассмотреть обфускацию исходного кода, в частности лексическую обфускацию. Исходя из определения обфускации, на вход функции *O* подается только один параметр – программа (исходный код программы). Однако в программном обеспечении, реализующем алгоритм обфускации, существуют параметры, которые зависят от разработчика или пользователя. Применительно к лексической обфускации алгоритм генерации новых имен переменных выбирается, как

правило, производителем инструмента обфускации и никакой смысловой нагрузки не несет. Предлагается использовать алгоритм переименования/генерации идентификаторов таким образом, чтобы обфусцированный код мог хранить ВЗ скрытым в именах классов, методов и переменных.

В качестве одного из подходов предлагается переименовывать идентификаторы, используя различный регистр таким образом, что символы верхнего регистра будут интерпретироваться как «0», а нижнего – как «1», образуя при этом последовательность, состоящую из «0» и «1», что по выше приведенному определению является ВЗ. На рис. 4 показан пример взаимного преобразования имени идентификатора и последовательности бит, которые необходимо выполнить при внедрении или извлечении ВЗ. Таким образом, переименовывая (генерируя) новые имена идентификаторов, можно сохранить строку размером до N бит, где N – сумма длин получившихся после переименования имен идентификаторов. С учетом того, что значение N для средних и больших программ достаточно велико, возможно использование избыточного кодирования или повторения ВЗ с целью его дополнительной защиты.

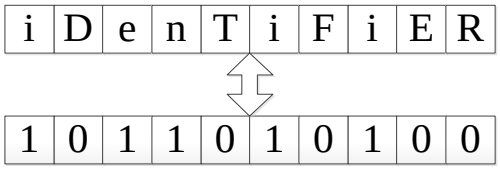

Рис. 4. Схема преобразования имени переменной в последовательность «0» и «1»

В качестве второго подхода использования лексической обфускации для внедрения ВЗ предлагается последовательно заменять первые символы имен идентификаторов на символы из ВЗ. К примеру, чтобы сохранить строку «copyright» имена идентификаторов могут быть следующими: «***c***or»,«***o***nio»,«***p***leg»,«***y***and»,«***r***ott» и т. д.

Описанные выше подходы имеют общую особенность: не требуется сохранение дополнительной информации о программе для внедрения и извлечения ВЗ – необходимы только алгоритм и программа-объект (исходный код).

Рассмотрим еще один алгоритм, который по своей природе схож с принципом использования решетки Кардано, которая применялась в 16–17 веках для передачи скрытых сообщений с использованием открытых текстов [11].

Для внедрения ВЗ используется строка *abcdef,* состоящая из 6 символов, которую будем называть несущей строкой и которая в последующем будет являться частью ключа для извлечения ВЗ. Для каждой буквы генерируется случайное значение, лежащее в промежутке от 2,6 до 5,0 (табл. 1), которое в дальнейшем будет соответствовать вероятности появления данной буквы в итоговом массиве. Границы 2,6 и 5,0 выбраны по принципу $k \pm 1{,}2$, где k – это вероятность появления символа в случайном наборе символов с алфавитом, состоящим из 26 символов (английский алфавит, который используется для именования идентификаторов), равная 3,8.

Таблица 1. Вероятность появления букв

Буква несущей строки	*a*	*b*	*c*	*d*	*e*	*f*
Вероятность	2,9	4,3	3,8	4,7	3,1	2,7

На основе табл. 1 строится таблица, указывающая позиции для сохранения копирайта строки внутри массива символов (табл. 2). Значения столбца «Коэффициенты» являются дополнительными множителями, которые позволяют определить позиции для записи символов ВЗ. Несущая строка совместно со значениями коэффициентов представляют собой ключ для извлечения ВЗ.

Таблица 2. Позиция для сохранения копирайта строки внутри массива символов

Несущая строка	*a*	*b*	*c*	*d*	*e*	*f*
Вероятности / Коэффициенты	2,9	4,3	3,8	4,7	3,1	2,7
10	29	43	38	47	31	27
110	319	473	418	517	341	297
210	609	903	798	987	651	567
310	899	1333	1178	1457	961	837
410	1189	1763	1558	1927	1271	1107
510	1479	2193	1938	2397	1581	1377
610	1769	2623	2318	2867	1891	1647
710	2059	3053	2698	3337	2201	1917
810	2349	3483	3078	3807	2511	2187
910	2639	3913	3458	4277	2821	2457
1010	2929	4343	3838	4747	3131	2727

Далее формируется итоговый массив, размерность которого сравнима с длиной всех имен идентификаторов, подвергаемых замене. При формировании итогового массива копирайт строка записывается на позиции, указанные в табл. 3. Остальные позиции итогового массива заполняются буквами английского алфавита (рис. 5) с учетом того, что символы несущей строки должны встречаться с вероятностью, указанной в табл. 1. К примеру, количество символов «*a*» в итоговом массиве должно быть равно 6000 · 0,029 = 174. Для повышения стойкости алгоритма необходимо, чтобы вероятность возникновения символов, не входящих в несущую строку, существенно не отклонялось от пределов 2,6 и 5,0.

Таблица 3. Позиция символа в массиве

Символы	Позиции
c	29
o	18
p	38
y	47
r	31
i	27
g	319
t	198
…	…

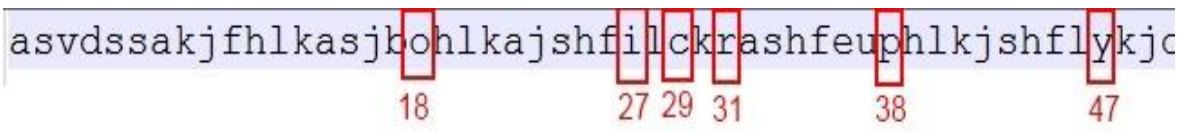

Рис. 5. Позиция символа копирайта строки в итоговом массиве

Для осуществления лексической обфускации над выбранным модулем итоговый массив разбивается на необходимое количество идентификаторов (длина каждого идентификатора выбирается произвольно) и производится последовательная замена каждого имени идентификатора модуля программы на строки, получившиеся в результате разбиения.

Для извлечения ВЗ необходимо получить все идентификаторы параметров модуля в порядке их следования и сформировать единый массив. Затем для несущей строки рассчитать вероятность появления каждого символа и с использованием коэффициентов (10, 110, 210, 310...) получить позиции символов копирайт строки, что является аналогом наложения решетки Кардано.

Изменение (увеличение) длины и содержания несущей строки, а также изменение коэффициентов дают возможность для дальнейшего изучения и совершенствования предложенного алгоритма.

Заключение

В работе рассмотрена возможность использования обфускации как инструмента для внедрения ВЗ в программный продукт с целью защиты интеллектуальной собственности. На примере лексической обфускации продемонстрированы подходы к внедрению ВЗ как с использованием дополнительного ключа, так и без него. Изучение данной темы в дальнейшем позволит усовершенствовать методы внедрения с помощью лексической обфускации и сделать их более устойчивыми к удалению и модификации. В качестве развития предложенной тематики может быть исследована возможность использования других видов обфускации для внедрения ВЗ.

LEXICAL OBFUSCATION AS A TOOL FOR WATERMARK EMBEDDING

Y.H. VASHINKO, A.V. BUDNIK, V.M. BONDARIK

Abstract

Given the definitions for obfuscation and watermark. Software that could implement obfuscation and watermarks was reviewed. New methods for embedding watermarks via obfuscation were proposed.

Список литературы

1. Таненбаум, Э. Компьютерные сети / Э. Таненбаум, Д. Уэзеролл. – 5-е изд. – СПб : Питер, 2012. – 960 с.

2. Олифер, Н. А. Основы компьютерных сетей / Н. А. Олифер, В. Г. Олифер. – СПб : Питер, 2009. – 352 с.

3. Eilam, E. Reversing: Secrets of Reverse Engineering / E. Eilam. – Wiley, 2005. – 619 с.

4. Fisher, C. Code Obfuscation Literature Survey / C. Fisher // University of Wisconsin; instruct. Madison. – 2005.

5. C# для профессионалов / под общ. ред. С. Рибсона. М. – 2012.

6. Афанасенко, А. Э. / Применение обфускации кода для защиты программного продукта // Материалы докл. и крат. сообщ. IV Бел.-рос. науч.-техн. конф. «Технические средства защиты информации», Минск – Нарочь, 29 мая – 2 июня 2006. / БГУИР ; редкол. : В. Ф. Голиков [и др.]. – Минск, 2006. – С. 42.

7. William Feng Zhu Concepts and Techniques in Software Watermarking and Obfuscation : A thesis submitted in partial fulfillment of the requirement of Doctor for Philosophy in Computer Science. MTI, 2007.

8. Halstead, M. Elements of Software Science (Operating and programming systems series) / M. Halstead. – New York : Elsevier North-Holland, 1977.

9. Kahn, D. The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet / D. Kahn. – New York : Simon and Schuster, 1996. – 1200 p.

10. Портянко, С. С. Статистический подход для внедрения водяных знаков в исполняемый код / С. С. Портянко, В. Н. Ярмолик // Докл. БГУИР. – 2005. – № 1. – С. 98–103.

11. Tamper-proofing Software Watermarks / C. Thomborson [et. al.]. // Proc. Second Australasian Information Security Workshop. ACS, CRPIT. – 2004. – Vol. 32. P. 27–36.