УДК 004.056.55

АНАЛИЗ И ВЫБОР МЕТОДОВ ЗАЩИТЫ ПРОГРАММНОГО ПРОДУКТА ОТ КОПИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ОБФУСКАЦИИ

Умяров Н.Х., Губенко Н.Е.
Донецкий национальный технический университет, Украина

В работе описаны два подхода к защите программного продукта от незаконного копирования – правовой и программный. Введено формальное определение запутывания программ, представлен краткий обзор основных методов обфускации, описаны достоинства и недостатки процесса обфускации.

1 Актуальность проблемы

В настоящее время защите кода программ от декомпиляции уделяется большое внимание, так как зачастую исходные тексты программ составляют коммерческую тайну. Кроме того, знание внутреннего устройства некоторых систем может помочь злоумышленникам организовать атаку на пользователей этой системы, используя при этом ошибки, допущенные при создании продукта.

2 Правовые аспекты защиты

Как отмечается в [1, 2], многие ресурсы позволяют зарегистрировать авторское право на программу. Владелец авторского права может подать в суд на нарушителя этих прав и выиграть судебный процесс, так как у него на руках есть доказательство авторства.

Авторское право – подотрасль гражданского права, регулирующая отношения, связанные с созданием и использованием (изданием, исполнением, показом и т. д.) произведений науки, литературы или искусства, то есть объективных результатов творческой деятельности людей в этих областях. Программы для ЭВМ и базы данных также охраняются авторским правом. Они приравнены к литературным произведениям и сборникам, соответственно [3].

Согласно гражданскому кодексу Украины [4], статье 433, п. 2, Компьютерные программы приобретают правовую охрану как литературные произведения, и такая правовая охрана распространяется на все виды программ, в том числе на прикладные программы, операционные системы, написанные на любом языке и в любой другой форме, включая исходный код. Исходный код компьютерной программы имеет черты письменного литературного произведения - код компьютерной программы может быть написан на разных языках программирования.

Согласно ст. 50 Закона Украины «Об авторском праве и смежных правах» нарушением авторского права, является:

- пиратство в сфере авторского права;
- плагиат – обнародование (опубликование), полностью или частично, чужой программы по именем лица, которое не является автором этого произведения;
- совершение действий, которые создают угрозу нарушения авторского права;
- подделка, изменение и извлечение информации, в частности в электронной форме, об управлении правами без разрешения субъектов авторского права или лица, которое осуществляет такое управление;

Таким образом, это дает позможность разработчику программного продукта подать иск в суд на нарушителя авторского права.

Однако следует отметить, что в связи со спецификой компьютерной программы, уровнем современного развития информационных технологий, нормы авторского права не могут защитить, например, от заимствования логических принципов, реализованных в этих программах, архитектуры построения программы и тому подобное, что порождает проблему воспроизведения функциональных

возможностей чужих компьютерных программ за короткий срок без нарушения международных соглашений и национального законодательства.

Таким образом, правовое законодательство является недостаточным регулятором защищенности. Поэтому, есть смысл использовать программную защиту, а именно, запутывание исходного кода программ с целью затруднить ее декомпиляцию и последующего копирования и распространения.

3 Программная защита исходного кода

Обфускация или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

Цели обфускации – оптимизация программы с целью уменьшения размера работающего кода и ускорения работы программы, демонстрация неочевидных возможностей языка программирования, затруднение декомпиляции/отладки проприетарных программ и т.д.

К недостаткам работы обфускаторов относятся [6]:

- Потеря гибкости кода. Код после обфускации может стать более зависимым от платформы или компилятора.
- Трудности отладки. Обфускатор не даёт постороннему выяснить, что делает код, но и не даёт разработчику отлаживать его.
- Ошибки в обфускаторах. Современный обфускатор – сложный программный комплекс. Есть ненулевая вероятность, что прошедший через обфускатор код вообще не будет работать. И чем сложнее разрабатываемая программа, тем больше эта вероятность.

Далее показан пример обфускации исходного кода Java-программы:

До обфускации:

```
package Main;
// Главный класс программы
public class HelloWorld {
        // Печать строки на экране
        public void print(String printStr) {
                System.out.println(printStr);
        }
        // Главная функция программы
        public static void main(String[] args) {
                new HelloWorld().print("Hello world");
        }
}
```

После обфускации:

```
package a;public class a{public void a(String a){System.out.println(a);}public static void main(String[] args){new a().a("Hello world")}}
```

4 Формальное опредление обфускации

1. Формальное опредление обфускации

Пусть П – множество всех программ (машин Тьюринга), удовлетворяющих сформулированным выше ограничениям, и пусть программа $p \in \Pi$ вычисляет функцию $f_p : Input \to Output$.

Подмножество $\pi \subseteq \Pi$ называется функциональным свойством если:

$$\forall p_1, p_2 \in \Pi(f_{p_1} = f_{p_1} \Rightarrow (p_1 \in \pi \Leftrightarrow p_2 \in \pi)) \quad (1)$$

Пусть π – функциональное свойство, $P \subseteq \Pi$ – класс программ такой, что существует

эффективная программа *c* такая, что для любой программы $p \subseteq P$:

$$c(p) = \begin{cases} 1, если\ p \in \pi \\ 0, иначе \end{cases} \quad (2)$$

Другими словами, для функционального свойства π мы определяем класс программ P таких, что существует эффективная программа-распознаватель *c* свойства π по программе *p* из класса P .

Вероятностная программа o называется запутывателем класса P относительно свойства $\pi(P,\pi)$ – запутывателем, если выполняются условия:

а) Эквивалентность преобразования запутывания. Для любой $p \in P$ и

$$\begin{aligned} & p' \in o(p) \\ & f_p = f_{p'} \\ & |p'| = poly(|p|), \\ & \forall x \in Dom_{f_0} \\ & time_{p'}(x) = poly(time(p(x))) \end{aligned} \quad (3)$$

Здесь y = poly(x) означает, что y ограничен полиномом некоторой степени от переменной $x, time_p(x)$ - время выполнения программы p на входе x , |p| - размер программы p.

б) Трудность определения свойств по запутанной программе. Для любого полинома *q* и для любой программы *a* такой, что a(o(P))={0,1}, и для любой выполняется time a(o(p)) = poly(|o(p)|), существует программа (вероятностная машина Тьюринга с оракулом) b , и при этом для любой $p \epsilon P$.

$$|Pr(a(o(p)) = c(p)) - Pr(b^p(1^{|p|}) = c(p))| \le \frac{1}{q(|p|)} \quad (4)$$

Вероятность определить свойство π по запутанной программе равна вероятности определения свойства π только по входам и выходам функции f_p. То есть, наличие текста запутанной программы ничего не даёт для выявления свойств этой программы.

Универсальный запутыватель - это программа *O*, которая для любого класса программ *P* и любого свойства π является (P, π)-запутывателем. Универсального запутывателя не существует. С практической точки зрения запутывание программы можно рассматривать как такое преобразование программы, которое делает её реинженеринг экономически невыгодным [7].

5 Выбор методов обфускации

Существует множество способов затруднить получение исходные текстов из бинарного файла. Подробно со всеми методами можно ознакомиться в работах [7, 8].

Первая группа методов – преобразования форматирования: удаление комментариев, переформатирование программы, удаление отладочной информации, изменение имён идентификаторов.

Вторая группа методов – преобразования потока управления программы – они изменяют граф потока управления одной функции, приводят к созданию новых функций.

Метод «Открытой вставки функций» заключается в том, что тело функции подставляется в точку вызова функции. Данное преобразование является стандартным для оптимизирующих компиляторов. Это преобразование одностороннее, то есть по преобразованной программе автоматически восстановить вставленные функции невозможно.

Метод «Вынос группы операторов». Данное преобразование является обратным к предыдущему и хорошо дополняет его. Некоторая группа операторов исходной программы выделяется в отдельную функцию. При необходимости создаются формальные параметры. Этот метод существенно затрудняет процесс декомпиляции программы.

Метод «Переплетение функции». Идея этого запутывающего преобразования в том, что две или

более функций объединяются в одну. Списки параметров исходных функций объединяются, и к ним добавляется ещё один параметр, который позволяет определить, какая функция в действительности выполняется.

Непрозрачные предикаты. Непрозрачные предикаты могут быть трёх видов: P^F – предикат, который всегда имеет значение «ложь», P^T – предикат, который всегда имеет значение «истина» и $P^?$ – предикат, который может принимать оба значения, и в момент запутывания текущее значение предиката известно.

Некоторые возможные способы введения непрозрачных предикатов и непрозрачных выражений:

1. Построение сложных булевских выражений с помощью эквивалентных преобразований. В простейшем случае мы можем взять k произвольных булевских переменных $x_1 \ldots x_k$ и построить из них тождество $(x_1 \cup \overline{x_1}) \cap \ldots \cap (x_k \cup \overline{x_k})$.

   Далее с помощью эквивалентных алгебраических преобразований часть скобок (или все) раскрываются, и в результате получается искомый непрозрачный предикат.
2. Использование комбинаторных тождеств, например

$$\sum_{i=1}^{n} C_n^i = 2^n .$$

3. Внесение недостижимого кода. Если в программу внесены непрозрачные предикаты видов P^F или P^T, ветки условия, соответствующие условию «истина» в первом случае и условию «ложь» во втором случае, никогда не будут выполняться. Фрагмент программы, который никогда не выполняется, называется недостижимым кодом.
4. Внесение избыточного кода. Избыточный код можно упростить или совсем удалить, так как вычисляется либо константное значение, либо значение, уже вычисленное ранее, наример создание избыточного кода из константы 256:

$$\sum_{i=1}^{8} C_8^i = 2^8 = 256 .$$

5. Преобразование сводимого графа потока управления к несводимому. Когда целевой язык (байт-код или машинный язык) более выразителен, чем исходный, можно использовать

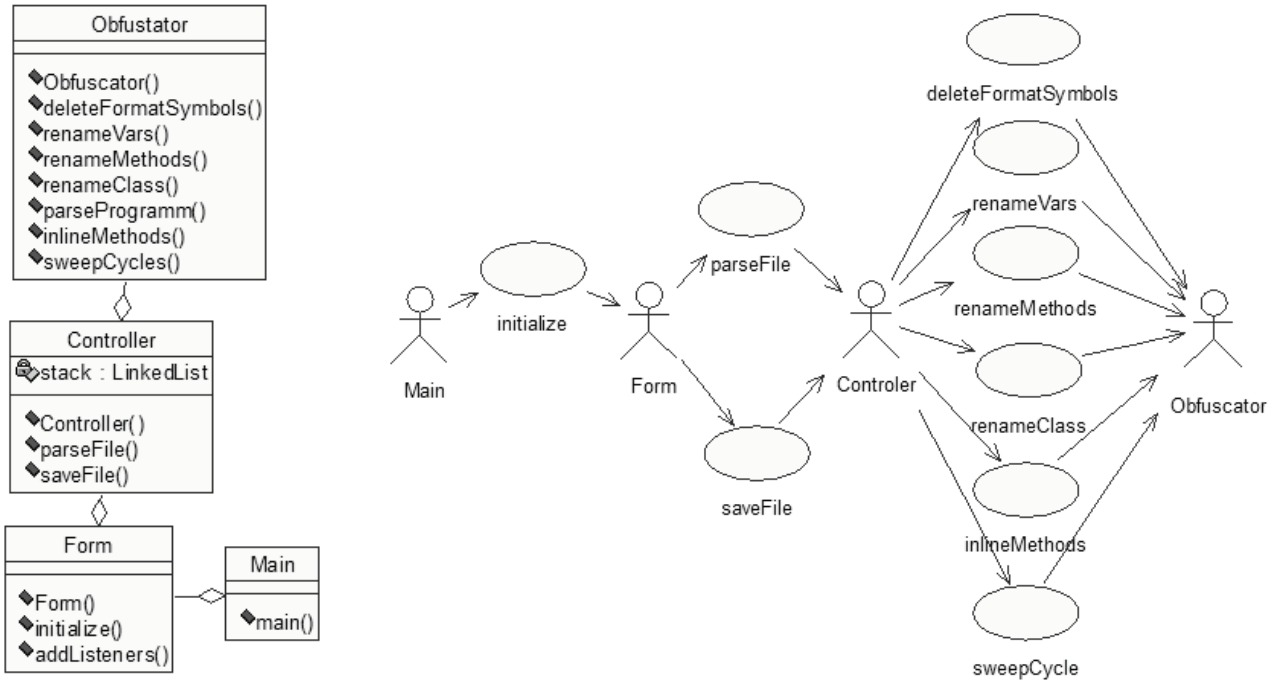

Рисунок 1. Логические UML модели системы “Обфускатор Java”:
а) диаграмма классов, б) диаграмма прецедентов

преобразования, «противоречащие» структуре исходного языка. В результате таких преобразований получаются последовательности инструкций целевого языка, не соответствующие ни одной из конструкций исходного языка.

Анализ перечисленных методов показал, что методы преобразования форматирования, открытой вставки функций, изменения имен локальных переменных, развертки циклов позволяют запутать программу, не приводя к существенному увеличению времени выполнения программного кода.

Результатом работы является объектно-ориентированная модель системы обфускации, изображенной на рис. 1.

Выводы

В тезисах представлены правовые и программные методы защиты программного продукта от копирования. Дано определение обфускации, построена математическая модель процесса обфускации, рассмотрены основные методы обфускации. Полученные результаты будут использоваться для создания программного продукта «Обфускатор Java».

Литература

[1] Регистрация авторских прав: http://ipstyle.ua/ru/services/copyright/copyright_registration.html

[2] Как зарегистрировать авторские права на веб-сайт? http://www.romanenko.ua/ru/library/article_site.html

[3] Хозяйственный кодекс Украины: http://zakon.rada.gov.ua/cgi-bin/laws/main.cgi?nreg=436-15

[4] Гражданский кодекс Украины: http://kodeksy.com.ua/ka/grajdanskij_kodeks_ukraini.htm

[5] Закон об авторском праве и смежных правах: http://www.medialaw.ru/exussrlaw/l/ua/copyright.htm

[6] Определение, основные цели и недостатки обфускации: http://ru.wikipedia.org/wiki/Обфускация

[7] Анализ запутывающих преобразований: http://citforum.ru/security/articles/analysis/

[8] Эссэ: методы защиты байт-кода Java и .NET: http://www.re.mipt.ru/infsec/2003/essay/2003_Byte-code_protection_in_Java_and_.NET__Babokin.pdf