УДК 003.26+347.78

В. А. Пласковицкий, студент (БГТУ); **П. П. Урбанович**, доктор технических наук, профессор, заведующий кафедрой информационных систем и технологий (БГТУ)

ЗАЩИТА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ И МОДИФИКАЦИИ МЕТОДАМИ ОБФУСКАЦИИ

Проанализированы особенности теоретической и практической реализации одного из методов защиты программного обеспечения – обфускации. Рассмотрены преимущества и недостатки различных подходов по использованию обфускации в качестве защиты от несанкционированного использования и модификации кода программы. В результате работы создано программное средство для реализации рассмотренных методов обфускации на уровне исходных и промежуточных кодов. Описаны особенности его интерфейса и функционал.

The features of the theoretical and practical implementation of one of the methods of protection software – obfuscation are analyzed. The advantages and disadvantages of different approaches to the use of obfuscation to protect of program code against unauthorized use and modification are considered. As a result of the work a software tool for the implementation of the considered methods of obfuscation the source-level and intermediate codes is created. The features of its interface and functionality are described.

Введение. Защита программного обеспечения (ПО) от несанкционированного использования в последнее время становится в ряд приоритетных задач в области информационных технологий и защиты авторских прав. В одних случаях приоритет отдается защите ПО от взлома, в других – кражи самого принципа работы ПО (или отдельных его модулей) с целью использования в конкурирующих компаниях. При этом доступность понимания кода программного продукта играет важнейшую роль в обоих случаях.

Известны различные методы решения указанной задачи. К числу эффективных относится обфускация (англ. obfuscation – запутывание) – метод защиты ПО, основанный на построении программного кода таким образом, чтобы его работа не изменилась на уровне выполнения, но стала более трудной для изучения [1].

Изменять, как и изучать, код можно на трех уровнях:

1) исходном – на котором разрабатывается ПО;

2) промежуточном – наборе инструкций, предназначенных для дополнительной обработки на виртуальной машине;

3) машинном – в который преобразуется программа после обработки компилятором для выполнения на ЭВМ.

Чем ниже уровень представления кода, тем сложнее его анализировать и тем эффективнее будут действовать добавленные туда препятствия. Но, безусловно, обфускация также требует понимания кода и от ее разработчиков. Поэтому реализация низкоуровневой обфускации требует специальных знаний.

С другой стороны, даже самая эффективная защита программного кода на машинном уровне будет бесполезна, если взломщику доступны исходные коды. Так, например, клиентская часть интернет-приложения, разработанная на JavaScript, передается пользователю в исходном виде.

Но важным является то, что благодаря распространению виртуальных машин (NET, Java, Flash), код приложений стал доступен на промежуточном уровне, который очень близок к исходному уровню, за тем исключением, что состоит из более универсальных наборов команд [2, 3]. Конечно, байт-код Java-машины прочесть сложнее, чем полностью восстановленный после декомпиляции код Flash [4], но и эта разница в сложности не является существенной проблемой.

Все это обусловливает гораздо большее значение обфускации на верхних уровнях кодирования [5]. Создание и описание одного из таких обфускаторов и стало целью данной работы.

Основная часть. Приемы обфускации могут быть самыми разными: от замены названий объектов до изменения потоков выполнения. Но в конечном счете все они сводятся к работе с текстом, в виде которого и представляется код. Под такой работой подразумевается поиск, анализ и замена отдельных фрагментов кода по заранее выработанным алгоритмам.

От того, какой инструмент будет использован для такой работы, зависят и все возможности обфускатора. Методов, объединенных общим названием или классом *String*, недостаточно, так как они не позволяют сформулировать действительно гибкое условие для поиска. Поэтому в основу нашей программы легли регулярные выражения (РВ) – наиболее мощный на сегодняшний день механизм описания искомых частей текста.

Набор возможностей РВ зависит от конкретной их реализации и может обладать дополнительными характеристиками, выходящими за классические рамки. Но ни одна из реализаций не может предоставить необходимой для

обфускации возможности замены текста. Поскольку замена в РВ опирается на текст, найденный в поисковом запросе, то это для многих методов обфускации не является достаточным. Например, метод замены имен объектов основан на составлении уникальных и нечитабельных названий, что невозможно осуществить, зная только их исходные значения (нужно использовать генератор имен, хранить список созданных значений и одним и тем же объектам присваивать одни и те же значения).

Этот означает, что обфускатор, помимо интерфейса для работы с РВ, должен обладать дополнительными возможностями настоящего языка программирования (ЯП).

На сегодняшний день существует достаточно много ЯП, обладающих хорошей поддержкой РВ на встроенном уровне: Perl, .NET, Java, AS 3.0 и др. Для языков, не обладающих такой поддержкой, разработаны специальные библиотеки. Поэтому выбор для этого конкретного ЯП в большой степени обусловлен личными вкусами разработчика. В данном случае был выбран C#, реализующий возможности библиотеки Framework. Но перевод большой части алгоритмов на другой ЯП не станет значительной сложностью, так как специфические возможности Framework и реализованные в нем механизмы работы с РВ намеренно используются только при действительной необходимости.

При разработке программы преследовалась реализация трех важных требований:

– создание удобного GUI для РВ;

– возможность использования приемов обфускации, выходящих за рамки РВ;

– возможность восстанавливать исходный вид кода после обфускации.

Текущий вид окна программы представлен на рис. 1.

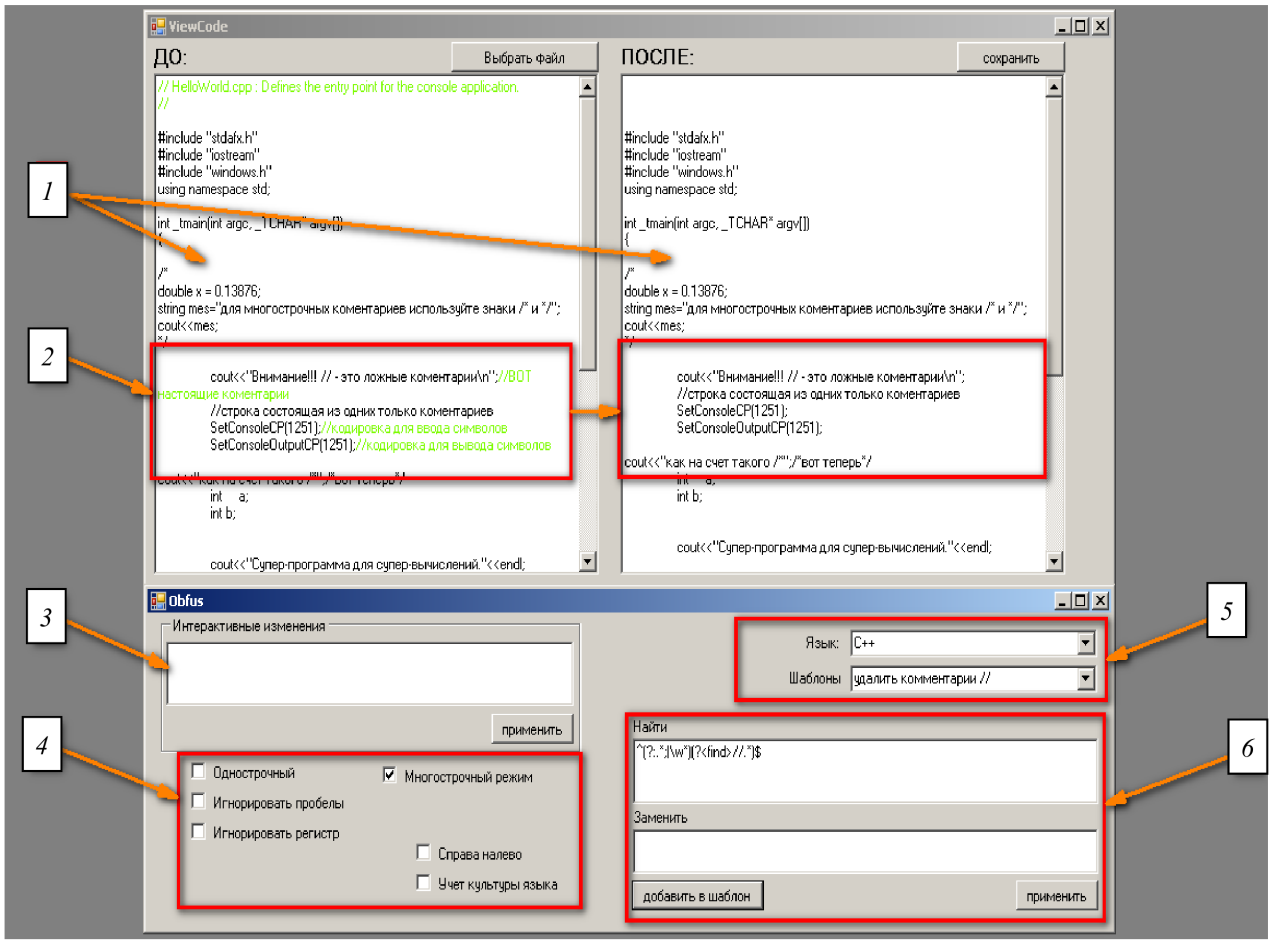

Рис. 1. Фрагмент работы программы «Обфускатор исходного кода»:
1 – текстовые окна для отображения кода до и после изменения; *2* – подсветка изменяемой части кода;
3 – поле для интерактивной замены; по нажатии на кнопку «применить» текст из этого поля вставляется в поле с исходным кодом (*1*) (если там был выделен текст, происходит вставка с заменой);
4 – настройка режима работы РВ; *5* – выбор готовых РВ (шаблонов); в списке шаблонов изначально уже располагаются наиболее характерные РВ для проведения обфускации (удаление комментариев, удаление перевода строки, запутывание переменных и т. д.), разработанные нами для различных языков программирования;
6 – поля для ввода собственных РВ, их применения и сохранения в виде шаблонов

Также была разработана расширенная версия редактора РВ (рис. 2).

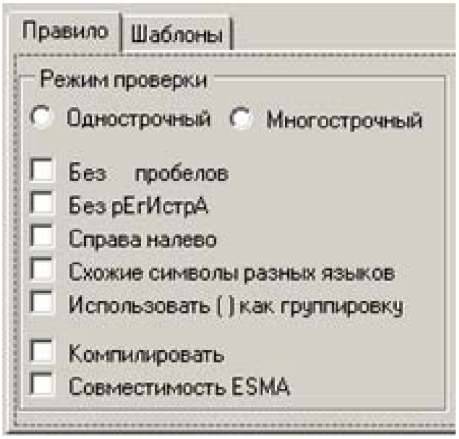

Рис. 2. Одно из окон расширенной версии редактора РВ

Отдельное внимание было уделено возможности одновременной обработки нескольких файлов и использованию фильтрации (прямой и инверсной). Это позволяет составлять набор желаемых для обфускации файлов как выбором вручную, так и указанием папок с записью расширений тех файлов, которые должны быть обработаны (или наоборот).

Все изменения, проводимые над кодом, фиксируются в специальном файле формата xml. Благодаря этому всегда можно восстановить исходный вид кода. Это правило распространяется и на ручной способ изменения, что позволяет использовать алгоритмы, реализация которых не представляется возможной либо не оправдывается сложностью.

Для того чтобы наиболее рационально осуществлять ручное редактирование, последнее проводится не прямой правкой интересующих участков текста, а их выделением и указанием желаемого результата.

Реализация правила хранения изменений заслуживает отдельного внимания, так как помимо того, что такое хранение должно предполагать возможность полного восстановления по нему исходных данных, оно должно запрещать взломщику возможность восстановления данных только на основе изучения этого файла.

Первое требование предусматривает хранение всех заменяемых символов, информации о месте их расположения в коде, а также информации о месте расположения символов замены для их удаления. Второе требование запрещает хранить наборы «строка до обфускации»/«строка после обфускации», поскольку достаточно будет просто извлечь из файла все строки, содержащие текст до обфускации. Ввиду того, что скрытие таких строк и является главной задачей при обфускации – весь смысл такой операции пропадает. Поэтому остается только один выход: сохранять информацию только о замененных и заменяющих символах.

Для замененных символов нужно хранить полный текст этих символов, строку, в которую они входят, и их позицию в строке.

Для заменяющих символов достаточно указать их длину. При этом строка вхождения и позиция в строке совпадают с соответствующими данными для заменяемого текста. В случае, когда в код вставляются лишние символы, для имитации их удаления необходимо длину заменяемого текста взять равной нулю, благодаря чему существенно сокращается объем файла (что очень важно в случае множественной обфускации).

Исходя из всего вышесказанного, предложена следующая структура кода:

```
<NOTE>
<BEGIN_TEXT number_string="Номер строки",
position="Начальная позиция в строке">
[!CDATA[
исходный текст
]]
</BEGIN_TEXT>
<END_TEXT length="Длина текста" />
</NOTE>
```

Или в сокращенном виде:

```
<N><B n="номер строки" p= "Начальная позиция в
строке">[!CDATA[текст]]</B><E l="длина тек-
ста"/></N>
```

В результате получается, что на каждую запись приходится 44 символа xml-разметки.

С учетом того, что файл нужно хранить в формате Unicode (для хранения любых символов), под каждый символ отводится 2 байта. Если предположить, что в среднем номер строки – 3-значное число, позиция в строке – 2-значное число и длина исходного и заменяющего текста составляет менее 100 символов (для заменяющего текста достаточно указать только длину, т. е. 2 символа), то получается 151 символ или около 300 байт на одну запись.

Это значит, что размер файла, хранящего 10 000 записей обфускации, не превысит 3 Мбайт (2,8 Мбайт), что удовлетворяет требованиям.

В качестве дополнительной возможности можно использовать функцию стека, благодаря чему гораздо проще создать правило для работы с многоуровневой вложенностью одинаковых (или различных) конструкций. Это особенно полезно не только для вложенных конструкций ЯП, но и при работе с любым другим

структурированным текстом, например, в xml- или html-формате.

Несмотря на схожесть конструкции многих языков, выработать универсальные правила довольно сложно. Так, например, удаление правил оформления невозможно применить для таких языков, как Phyton, поскольку в нем на табуляции основана обработка вложенных конструкций. Поэтому в программу добавлен фильтр правил РВ по различным ЯП. Это позволяет разрабатывать методы обфускации для конкретных языков, ограничив их использование, и не обращать при этом на них внимание при работе с другими ЯП.

При разработке правил крайне важно уделить внимание спецификации языков. Даже там, где пробельные элементы не влияют на основной код, сокращение переносов строк в заголовках подключаемых библиотек либо в других объявлениях может быть недопустимо.

Заключение. К числу новых методов защиты ПО относят такие, которые основаны на применении методов обфускации (а также водяных знаков и отпечатков пальцев). Данные технологии пока не получили такого распространения, как традиционные. Эта особенность использования связана во многом с тем, что многие теоретические и практические аспекты их реализации пока слабо изучены.

Результаты, описанные в данной статье, в основном касаются создания и использования разработанной авторской программы, реализующей возможности практического исследования методов обфускации, направленных на максимальное затруднение другим лицам осуществлять операции так называемого обратного проектирования.

Часть задуманных возможностей, в силу большого объема кодирования, еще не реализована в полной мере, часть – нуждается в дополнительной отладке и исследованиях. Тем не менее программа пригодна для практического использования. При этом помимо обфускации, ее можно применять для обработки поддающегося описанию текста, например, html/xml-файлов.

В качестве одного из важных положительных практических результатов можно отметить то, что программное средство выявило широкие возможности при очистке веб-страниц от лишнего мусора, получаемого в результате конвертации их из различных форматов (doc, pdf и др.). Программа также может успешно применяться для сжатия объема кода за счет сокращения числа операций форматирования и замены названий объектов на более короткие (важный фактор для интернета).

Будучи относительно новой технологией защиты, обфускация уже сегодня нашла широкое распространение во всех сферах программирования. Но важно понимать, что ни один из методов обфускации не удаляет из программы логику, а лишь затрудняет ее понимание. Поэтому надежность такой защиты существенно зависит не только от создателя кода, но и уровня знаний взломщика.

Литература

1. Ярмолик, В. Н. Криптография, стеганография и охрана авторского права / В. Н. Ярмолик, С. С. Портянко, С. В. Ярмолик. – Минск: Изд. центр БГУ, 2007. – 240 с.

2. Лившиц, Ю. Запутывание (обфускация) программ / Ю. Лившиц [Электронный ресурс]. – 2011. – Режим доступа: http://logic.pdmi.ras.ru/~yura/apers/lifshits2005obfuscation. – Дата доступа: 24.02.2011.

3. Понаморев, А. Защита Net продуктов от подглядывания и воспроизведения / А. Понаморев [Электронный ресурс]. – 2011. – Режим доступа: http://www.aspnetmania.com/Articles/Article/33.html. – Дата доступа: 25.02.2011.

4. Защита flash-объектов от декомпиляции [Электронный ресурс]. – 2011. – Режим доступа: http://flash-world.ru/flash_protection. – Дата доступа: 25.02.2011.

5. Касперский, К. Обфускация и ее преодоление / К. Касперский [Электронный ресурс]. – 2011. – Режим доступа: http://www.insidepro.com/kk/080/080r.shtml. – Дата доступа: 25.02.2011.

Поступила 28.02.2011