Доклады БГУИР
2004 *январь–март* *№ 1*

УДК 681.3.06, 681.324.06

СОВРЕМЕННЫЕ МЕТОДЫ И СРЕДСТВА ЗАЩИТЫ АВТОРСКИХ ПРАВ РАЗРАБОТЧИКОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ*

В.Н. ЯРМОЛИК, С.С. ПОРТЯНКО

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 19 ноября 2003

В данной статье приведен обзор существующих подходов к борьбе с неправомерными действиями, связанными с нарушением авторских прав на программное обеспечение. Рассмотрены области применения, достоинства и недостатки существующих методов. Также приведены результаты экспериментальных исследований программных модулей, которые проводились с целью выявления таких свойств программного обеспечения, которые открывают возможность для проектирования новых технологий защиты авторских прав, в частности, основанных на применении цифровых водяных знаков. В последней части статьи предложен ряд алгоритмов внедрения водяного знака в программный модуль, базирующихся на результатах проведенных статистических исследований, и определено направление дальнейшей работы.

Ключевые слова: авторские права, водяной знак, защита.

Введение

Защита программных продуктов от несанкционированного использования является первостепенной задачей для их разработчиков не только потому, что количество нелегальных копий продуктов ведущих форм достигает от 30 до 98% в зависимости от региона и страны [1], но и потому, что возникли новые более совершенные угрозы, чем простое нелегальное копирование программных продуктов. Выделяют три основных категории атак на современное программное обеспечение (ПО): нелегальное использование или воровство ПО — "Software Piracy"; обратное проектирование — "Reverse Engineering", когда из ПО извлекаются наиболее ценные модули или части исходного кода, и модификация исходного кода — "Software Tampering", приводящая к существенным потерям разработчиков ПО вплоть до потери ими авторских прав [2–4].

Первые попытки разработчиков ПО по защите авторских прав заключались в использовании различных форм технической защиты от копирования программ. Предлагалась привязка ПО с конкретной ЭВМ или к дополнительным аппаратным модулям, пластиковой картой, жестким диском или CD, без физического наличия которых инициализация и исполнение функций конкретным защищаемым ПО оказывалось невозможным [5, 6]. Очень популярным сегодня является электронный ключ — "Software Dongle", который является достаточно сложным микроэлектронным устройством, подключаемым к одному из портов ЭВМ. Только при его наличии оказывается возможным использование защищаемой данным конкретным ключом некоторой программы. В процессе инициализации и исполнения защищаемая программа постоянно

* Работа выполнялась при поддержке гранта BMBF BLR 02/006.

обменивается информацией с ключом. Невозможность физического копирования ключа и его высокая криптостойкость, задаваемая криптографическими алгоритмами, реализуемыми аппаратными средствами ключа, предопределили их широкое использование. Технология защиты ПО с применением электронных ключей заключается в использовании уникальных идентификаторов, которые хранятся в недоступной для чтения пользователями содержимого памяти ключей. В качестве идентификаторов (криптографических ключей) используются уникальные многоразрядные псевдослучайные коды, а также информация, характеризующая особенности защищаемого продукта, его версию, конфигурацию, срок действия лицензии, а также данные о разработчиках ПО и конкретного пользователя, легально купившего ПО. Основные недостатки защиты ПО с помощью электронных ключей связаны с ухудшением технических характеристик как защищаемого приложения, так и вычислительной среды, в которой ПО реализуется. Самым уязвимым элементом рассматриваемой технологии защиты ПО является очевидность функций, реализуемых в защищаемом приложении и отвечающих за обмен информации между ПО и электронным ключом. Это в конечном итоге существенно увеличивает уязвимость ПО различным атакам, направленным на нелегальное использование интеллектуальной собственности, представленной в виде ПО.

Попытка решить проблему защиты авторских прав на ПО без использования аппаратных решений привели к интенсивному развитию новых подходов, основанных на использовании последних достижений криптографии и стеганографии [7–9].

Современные методы защиты программного обеспечения

Защита ПО от нелегального использования его отдельных функциональных модулей в последние годы все более чаще достигается путем обфусцирования (obfuscation) исходного кода. Под обфусцированием понимается преобразование кода программы так, что бы максимально затруднить процесс декомпиляции и анализа кода с целью восстановления реализованных в нем алгоритмов [2-4,10]. Теоретической основой обфусцирования является эквивалентное преобразование алгоритмов, которое также используется для реализации современных средств оптимизации ПО. Обфускаторы преобразуют защищаемое ПО таким образом, что трансформированная программа является функционально эквивалентной исходной программе, однако ее код становиться чрезвычайно сложным и нечитаемым для его понимания, что должно предотвратить атаки типа обратное проектирование. Формально обфусцирование состоит в отображении исходной программы $P=\{\ldots,S_j,\ldots\}$, состоящей из исходного кода объектов S_j, в новую программу $P^*=\{\ldots,T_i(S_j),\ldots\}$, на основании эквивалентных преобразований T_i. Простейшим примером подобного преобразования может служить фрагмент кода, представленного на рис. 1.

```
i=1;
while (i<100) {
        …A[i]…;
        i++;
}
```

Рис. 1. Фрагмент кода перед его обфусцированием

В примере реализован цикл, в котором обрабатываются элементы одномерного массива $A[i]$.

В качестве эквивалентных преобразований используем преобразование целочисленных данных путем замены переменной i на переменную $i^*=8\times i+3$ и введем проверку условия $j^2(j+1)^2=0 \bmod 4$, которое выполняется для любых целых j. Однако данный факт не будет очевидным для лица, занимающегося анализом исходного кода алгоритма. В результате получим код, показанный на рис.2.

```
i=11; j=37;
while ((i<803) && (j*j*(j+1)*(j+1)%4==0)) {
        … A[(i-3)/8]…;
        i+=8;
        j=j*i+7;
}
```

Рис.2. Фрагмент кода после его обфусцирования

Несмотря на очевидную эффективность методов обфусцирования, следует отметить их определяющий недостаток, заключающийся, как правило, в увеличении размера исходного кода и времени его исполнения.

Существует большое множество критических приложений, для которых исполнение модифицированного кода является недопустимым и приводящим к значительному ущербу, в том числе и материальному [10]. Это в первую очередь относится к электронной коммерции, различным финансовым приложениям и различного рода данным, хранимым в памяти ЭВМ. С целью защиты программного приложения от модификации используются методы защиты кодов программ от модификаций ("temper-proofing") [11, 12]. Основной функцией подобных методов является функция обнаружения внесенных изменений в код программы перед его инициализацией и выполнением. Целостность исходного кода может быть определена на основании сигнатур ("signature") [13, 14] или контрольных сумм, полученных на базе однонаправленных функций, например SHA [15, 16] или сигнатурного анализатора [17]. Более совершенные подходы основаны на теории и практике самотестирования кода приложения либо используемых им данных [18, 19].

Наибольший интерес в настоящее время привлекают методы защиты ПО на базе водяных знаков ("watermarking") и отпечатков пальцев ("fingerprinting"), которые заключаются во внедрении в ПО информации об авторских правах на этот продукт [20]. В случае отпечатков пальцев в ПО дополнительно внедряется информация о легальном покупателе. Например, отпечатки пальцев могут содержать биты, идентифицирующие покупателя, продавца и само ПО. Внедряемая информация должна быть устойчива к различным преобразованиям, например, таким, как обфусцирование и декомпиляция.

Существует большое множество различных систем водяных знаков. Наиболее простым из них являются системы, основанные на внедрении необходимой информации в комментарии в теле программы — /* Software Company Copyrights, в виде константы CONST C="Software Company Copyrights" или совокупности значений переменной

```
switch e {
case 1: V=’S’
        case 2: V=’o’
        case 3: V=’f’
        case 4: V=’t’
        …
```

Приведенные примеры относятся к так называемым статическим системам водяных знаков и являются достаточно уязвимыми для различных атак.

В [21] описывается метод, относящийся к статическим водяным знакам, позволяющий внедрить водяной знак в тело Java-программы еще на этапе написания ее исходного кода. Предлагается использовать так называемый фиктивный код (последовательность операций, которая никогда не выполняется в силу того, что ей предшествует никогда не выполняемый условный оператор). Водяной знак встраивается прямо в исходный код приложения для обеспечения разработчику возможности подобрать "липовый" код таким образом, чтобы он выглядел максимально правдоподобно. Разработчики полагаются на участие человека в этом процессе, поэтому внедрение водяного знака не может быть полностью автоматизировано. Авторы данного решения полагаются на то, что в качестве атаки против водяного знака будут использованы обфускаторы, воздействующие на имена переменных и методов, лишая их смысла и оставляя при этом сам байт-код программы неизменным.

Технология внедрения водяного знака в программу, основанная на статистических свойствах исполняемого кода, была предложена в [22]. Предлагается путем незначительных изменений частот встречаемости ассемблерных команд осуществлять кодирование дополнительной информации. Эти изменения достигаются путем применения эквивалентных преобразований кода. Главной сложностью является необходимость построения достаточно большого списка операций преобразования машинного кода, не влияющих на логику работы программы и не оказывающих существенное влияния на такие параметры, как размер программы и время ее выполнения.

Определенный интерес представляют собой системы динамических водяных знаков [23]. Главное отличие динамических водяных знаков от статических в том, что они проявляют себя только в результате запуска программы на выполнение с некоторым нетипичным для нее набором входных данных и имеют название "Ester Egg". Обычно результатом работы приложения для нетипичных входных данных является отображение на экране сообщения об авторских правах.

Основной проблемой при внедрении водяных знаков в ПО является необходимость использования таких преобразований, которые не изменяли бы стандартные свойства оригинальных программ и не ухудшали такие их характеристики, как размер исходного кода и время выполнения. В то же время сохранение основных статистических характеристик и использование их для реализации водяных знаков представляется весьма перспективным направлением исследований.

Исследование статистических свойств исполняемого кода

Практически любое ПО (далее для простоты "программа") состоит из двух основных составляющих — внутренних данных (расположенных в сегментах данных) и исполняемого кода. Природа внутренних данных программы и их статистические свойства сильно зависят от функций программы и реализованных в ней алгоритмов. Для исполняемого кода программы проявляется ряд закономерностей, сохраняющихся от программы к программе. Эти закономерности обусловлены тем, что исполняемый код приложения представляет собой последовательность из определенного ограниченного множества машинных инструкций. Причем порядок следования большинства из них, а так же частоты использования как отдельных машинных команд, так и определенных конструкций, образованных двумя и более командами, определяется такими параметрами, как архитектура среды выполнения, синтаксическая корректность, требования к эффективности. По этой причине сегмент кода каждого приложения обладает рядом статистических характеристик, сохраняющихся от программы к программе и практически не зависящих от реализованной в ней функциональности [23]. К ним можно отнести такие параметры, как вероятность появления той или иной инструкции, вероятность появления определенной инструкции в указанной точке программы, вероятность следования рассматриваемой инструкции за некоторой другой, среднее расстояние между повторяющимися инструкциями в программе и др. Например, как видно на рис. 3, среднее значение номера команды, использованной в программе ($M(x)$), практически не зависит от самой программы и ее длины.

Измерения проводились для более чем 20 программных модулей (.exe, .dll) прикладного и системного назначения. Исследуемая закономерность наблюдается и при осуществлении вычисления величины $M(x)$ не для всех команд модуля (N), а для некоторого выбранного случайно их подмножества из L команд. На рис. 4 приведены аналогичные зависимости для $L=N$, $L=N/10$ и $L=N/100$. Из полученных результатов можно сделать вывод о том, что и при осуществлении выборки подмножества команд из программы величина параметра $M(x)$ не выходит за пределы определенного интервала значений, и, как показали эксперименты, для $L=N/100$ $M(x)$ находится в интервале (55, 95).

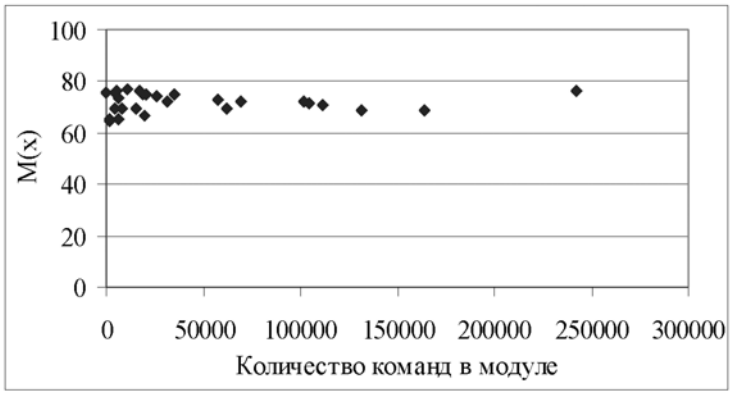

Рис. 3. Зависимость *M(x)* от числа команд в модуле

Рис. 4. Зависимость *M(x)* от количества команд в модуле при выборках 100, 10 и 1%

Еще одной статистической характеристикой исполняемого кода является среднее расстояние между одинаковыми командами. Данный параметр характеризуется функцией *S(k)*, значения которой равняются количеству совпавших команд в оригинальной программе *P* и программе *P'*, являющейся циклически сдвинутой на *k* позиций *P*. Под совпавшими командами понимаются стоящие на одной и той же позиции в *P* и *P'* одинаковые команды. На рис. 5 показана полученная экспериментально зависимость числа совпадений команд *S(k)* от величины сдвига *k* для значений *k* в интервале 1–100.

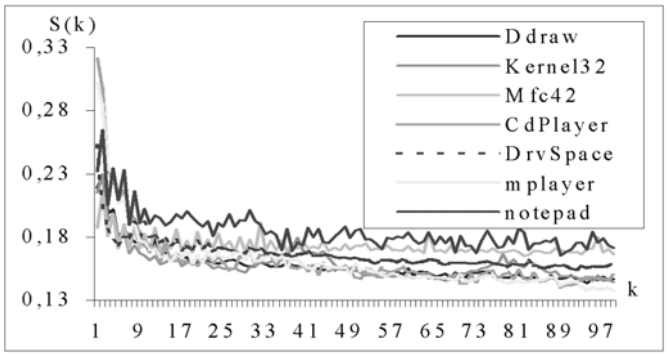

Рис. 5. Экспериментальная зависимость *S(k)*

В результате проведенных экспериментальных исследований данного показателя обнаружилось, что начиная с некоторого значения *k* величина *S(k)* не выходит за пределы фиксированного интервала.

Значение рассмотренного параметра может служить одной из статистических характеристик программы, которая не зависит от ее содержания, поскольку, как показали результаты экспериментов, начиная с некоторого значения *k* (порядка 2000 и выше) значение величины

$S(k)$ не выходит за пределы определенного интервала, на границы которого оказывают лишь незначительное влияние реализованные в программе алгоритмы.

Предлагаемые методы внедрения водяных знаков

Внедрение признака авторства при помощи искажений частотных свойств программы. Приведенные выше результаты экспериментов показывают, что некоторые статистические характеристики кода программы на языке ассемблера в очень незначительной мере зависят от функциональности программы. К ним относятся такие характеристики, как частоты встречаемости различных команд. Наличие данной особенности в исполняемом коде открывает возможность для внесения в него некоторого дополнительного уникального признака, отличающего программу от множества всех остальных. Данным признаком может являться отклонение статистической характеристики от стандартного или типичного ее вида. Например, нам известно, что частота использования команды *call* в теле произвольной программы равняется некоторой величине f_{call}, тогда, преобразовав программу таким образом, что бы данная команда встречалась гораздо чаще ($f'_{call} >> f_{call}$) либо гораздо реже ($f'_{call} << f_{call}$), мы внесем в программу признак, отличающий ее от всех остальных, который может быть рассмотрен как водяной знак. Подобный подход является достаточно тривиальным и часто используемым решением. В качестве основы всех нижеприведенных методов внедрения водяных знаков мы будем использовать изменения в одной из статистических характеристик программы. При этом для построения данных характеристик будут использованы лишь те параметры, значения которых являются константными либо меняющимися в заранее известном диапазоне не только для программы, в которую требуется поместить водяной знак, но и для всего множества программ, к которому она принадлежит. Преднамеренно внося изменения в статистическую характеристику программы, мы наделяем ее код отличительной особенностью, наличие которой позволит в будущем утверждать, что она могла быть создана только тем, кто данную особенность привнес.

Изменение значения среднего номера встреченной команды. Аналогичным образом можно воздействовать не только на частоту встречаемости одной команды, но и нескольких команд [23], и в этом случае параметром, резкое отклонение в значении которого от стандартного будет говорить о наличии в программе водяного знака, будет средний номер выбранной из программы ассемблерной команды M_x. Для того чтобы программа не только обладала рассматриваемым признаком, но и его наличие нельзя было обнаружить, не располагая некоторыми секретными данными, например, ключом, можно поступить следующим образом. Известно, что значение такого параметра, как M_x, располагается в некотором интервале, границы которого фиксированы, что было подтверждено экспериментально (рис. 3). Таким образом, изменение величины данного параметра в пределах этого интервала не окажет существенного влияния на частотные характеристики кода. Пусть программа P состоит из N ассемблерных команд. Исходя из результатов проведенных экспериментов, можно сделать предположение о том, что для любой программы для любых случайно выбранных из нее L ($L<N$) команд величина такого параметра M_x также будет заключена в определенном интервале (см. рис. 4). Можно подобрать такой вид воздействия теперь уже не на всю программу, а на выбранные из нее L команд, который приведет к отклонению рассматриваемого параметра далеко за пределы интервала его возможных значений в большую или меньшую сторону, не приводящих, однако, к выходу аналогичного параметра для всей программы за границы экспериментально определенного интервала. Подобный подход был предложен в [24] для внедрения водяного знака в графические изображения.

В качестве воздействия на программу, приводящего к требуемым изменениям в частотах встречаемости выбранных команд, необходимо использовать эквивалентные преобразования ассемблерного кода. Эквивалентные преобразования заключаются в замене одной команды либо группы команд на другие. Замена должна осуществляться таким образом, чтобы полученный в результате код был работоспособен и логика работы программы не менялась. Возможность осуществления подобных преобразований обусловливается свойством языка ассемблера, в котором выполнение одних и тех же действий возможно с использованием различных команд.

Сдвиг распределения разностей номеров двух последовательно выбранных команд. Рассмотрим еще одно свойство исполняемого кода, которое может быть использовано для внедрения в программу скрытого признака. Если некоторая случайная величина (СВ) *x* распределена равномерно, то в соответствии с центральной предельной теоремой [25] величина

$$y_k = \sum_{i=1}^{n} x_i \qquad (1)$$

где x_i — равномерно распределенная случайная величина, n — количество суммируемых значений, обладает нормальным распределением. В случае если СВ *x* распределена равномерно в интервале от $-D$ до D, то математическое ожидание (МО) СВ *y* будет равным 0.

Если же равномерно распределенная СВ распределена в интервале от 0 до B, то последовательность значений

$$z_k = x_{2i} - x_{2i+1}, \qquad (2)$$

будет также обладать нормальным распределением с МО $M(z_k)=0$.

В исполняемом коде программы в качестве случайной величины можно рассматривать номер выбранной ассемблерной команды. Для того чтобы данная величина обладала описанным выше свойством, необходимо, что бы распределение ее было близким к равномерному. Однако распределение номеров отдельных ассемблерных инструкций этому условию не удовлетворяет.

Данному условию удовлетворяет величина, равная номеру группы в списке групп из нескольких команд, сформированных особым образом. Группы формируются так, чтобы частоты их встречаемости в коде программы были максимально близкими.

Примеры сформированных групп команд приведены в табл. 3.

Таблица 3. **Примеры близких по частоте встречаемости группы команд**

Группа	Частота встречаемости
call, mov, mov	0,0016245241
call, or, mov, mov	0,0014757310
cmp, jbe	0,0015128968
je,and	0,0016741071

На рис. 6 показана гистограмма распределения частот встречаемости групп, сформированных с учетом упомянутого условия.

Для сформированного множества близких по частоте встречаемости групп команд распределение разностей вида (2), где x_{2i}, x_{2i+1} — номера (в упорядоченном списке групп инструкций) двух выбранных друг за другом из программы групп команд, имеет следующий вид (рис. 7).

Как видно из рисунка, благодаря тому, что были подобраны группы команд с наиболее близкой частотой встречаемости, полученное распределение разностей (2) имеет вид, близкий к нормальному. Случайная величина равная номеру группы команд в упорядоченном списке групп обладает свойством, позволяющим произвести адаптацию метода Pathwork [23] для внедрения водяного знака в исполняемый код программы. Способ внедрения будет заключаться в воздействии на частоты встречаемости сформированных групп команд таким образом, чтобы МО величины z_i было отличным от 0. Наличие такой особенности и будет свидетельствовать о присутствии в программе водяного знака. Для того чтобы водяной знак нельзя было обнаружить, не имея секретного ключа, можно использовать подход, аналогичный вышеизложенному — добиваться отклонения МО величины z_i не для всей программы, а только для подмножества ее команд, позиции которых могут быть получены только с помощью генератора псевдослучайных чисел инициализированного секретным ключом A_0.

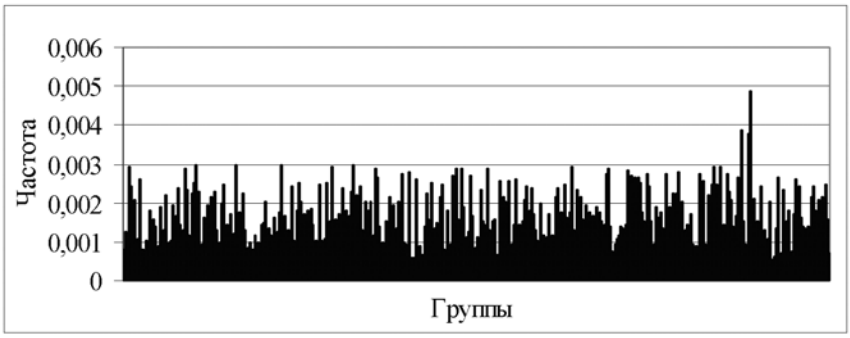

Рис. 6. Распределение частот встречаемости групп содержащих переменное число команд

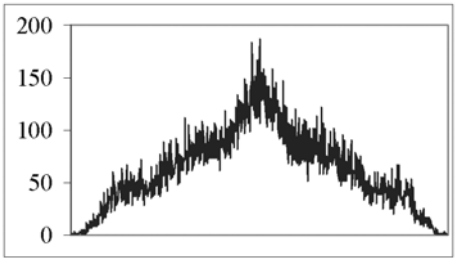

Рис. 7. Полученное экспериментально распределение величины z_k

Кодирование последовательности символов, образующих водяной знак (ВЗ)

Пусть $S=\{S_1, S_2, ..., S_m\}$ - двоичная последовательность, которой представлен водяной знак.

Необходимо разместить последовательность S в теле программы P таким образом, что бы:

а) модифицированная программа P', содержащая S, имела такое же наблюдаемое поведение, что и P;

б) последовательность S невозможно было извлечь, не располагая некоторым секретным ключем A_0;

в) способ размещения S в P определяется значением ключа A_0 и использует некоторое его свойство, исключающее возможность случайного появления S в теле программы.

Использование команд-синонимов. Для внедрения последовательности символов в код программы можно использовать то же свойство ассемблерного кода, которое применялось для осуществления эквивалентных преобразований. Таким образом, выбирая, каким именно из возможных способов в рассматриваемом участке программы будет выполняться та или иная операция, можно производить кодирование двоичных чисел. Например, обнуление содержимого регистра eax при помощи команды *mov eax,*0 будет кодировать ‘1’, а обнуление этого же регистра при помощи *xor eax,eax* — ‘0’.

Модифицирование адресов передачи управления. В коде программы такие команды, как *call, je, jne, jmp,* (команды передачи управления) встречаются достаточно часто. Управление может передаваться как на четное количество байт вперед или назад, так и на нечетное. Сопоставив переходам в теле программы на четное количество байт "0", а переходам на нечетное количество – "1", мы получим бинарную последовательность, вероятность встречи нуля и единицы в которой будет очень близкой. Это объясняется тем, что в программе переходы на четное и нечетное количество байт встречаются одинаково часто, что было подтверждено результатами проведенных экспериментальных исследований. Последовательность бит можно закодировать путем изменения длин последовательно идущих переходов. Эти изменения могут быть достигнуты либо с помощью вставки никогда не выполняющихся команд, либо данных.

Внедрение признака авторства путем внесения изменений в автокорреляционную характеристику. Как уже отмечалось выше, автокорреляционная зависимость программы об-

ладает свойством, заключающимся в том, что процент совпадений в оригинальной *P* и циклически сдвинутой программе *P'* начиная с некоторого значения сдвига становится постоянным, очень несущественно меняющимся от программы к программе. Эксперименты показали, что для всего множества исследованных программ значения параметра *S(k)* (количество совпадений при сдвиге на *k* позиций) не выходили за пределы интервала от 12 до 22% для сдвигов на число позиций, равное 100 и более. Отклонения в виде данной автокорреляционной функции от ее стандартного вида могут быть использованы в качестве средства, подтверждающего наличие водяного знака. Например, можно преобразовать программу таким образом, что для некоторых значений величины сдвига *k* количество совпадающих команд будет существенно ниже либо выше стандартного. Идея использования данной характеристики исполняемого кода может быть применена для внедрения водяного знака [26]. Например, возможен следующий алгоритм его постановки.

1. Пусть *W* — код, являющийся водяным знаком. Используя данное число в качестве начального значения, для генератора псевдослучайных чисел генерируются числа $A_1, A_2, \ldots, A_m$.

2. В программу *P* в позиции с номерами $A_1, A_2, \ldots, A_m$ вставляется пустая команда *nop*.

3. Производятся эквивалентные преобразования кода программы *P*, оставляя без изменений вставленные команды *nop* с целью достижения обращения функции *S(k)* в "0" на некотором множестве $k_1, k_2, \ldots k_t$ без учета совпадений в *nop*.

4. Вставленные ранее команды *nop* удаляются, программа не меняет логики своей работы, однако достигнутая на предыдущем шаге уникальная статистическая характеристика нарушается.

Полученная в результате шагов 1–4 программа *P''* содержит отличающий ее от всего множества других программ признак, являющийся водяным знаком. Порядок действий для проверки наличия водяного знака в программе следующий:

1. Разработчик, имея *W*, генерирует на его основе последовательность $A_1, A_2, \ldots, A_m$.

2. По адресам $A_1, A_2, \ldots, A_m$ в *P''* вставляются команды *nop*.

3. Строится зависимость *S(t)*, в которой для $k = k_1, k_2, \ldots k_t$ совпадения присутствуют только для команды *nop*, числа $k_1, k_2, \ldots k_t$ также могут быть одной из компонент водяного знака, кодируя, например, фамилию разработчика.

Заключение

Дальнейшие исследования в области внедрения водяных знаков в ПО будут направлены на практическую реализацию предложенных алгоритмов внедрения, а также определения границ применимости подобных методик (класс приложений, размер исполняемых модулей, максимальный объем внедряемых данных, язык, на котором написана программа) и устойчивости разработанных методов против возможных предпринимаемых с целью противостояния водяным знакам атак.

PRESENT-DAY TECHNIQUES AND MEANS FOR SOFTWARE DEVELOPERS AUTHIRSHIP PROTECTION

V.N. YARMOLIK, S.S. PARTSIANKA

Abstract

In this paper addicted a review of existing approaches to opposition with wrongful actions, related to software authorship violation. Considered fields of application, merits and drawbacks of existing techniques. Here also addicted results of software modules experimental studies, which were performed with the goal of educing such properties of software, which open possibility for new authorship protection technologies designing, particularly, based on digital watermarks application. In the

last part of this paper proposed a number of algorithms of watermark embedding into software module, based on results of performed statistical studies, and determined the direction of further work.

Литература

1. Business Software Alliance. The cost of software piracy: BSA's glabal enforcement policy. http://www.rad.net.id/bsa/piracy/globalfact.html, 1996.
2. *Yarmolik V.N., Portyanko S.S.* State of the ART in Software Ownership Protection // Computer Information Systems and Industrial Management Application / Editors: Khalid Saeed, Romuald Mosdorf, Olli-Pekka Hilmola. Bialystok, Poland, 2003. P. 188-195.
3. *Collberg C.S., Thomborson C.* Watermarking, Tamper-proofing and Obfuscation – Tools for Software protection // Computer Science Technical Report #170. University of Auckland, Aucland, New Zealand. February 10, 2000.
4. *Charbon E., Torunoglu I.H.* On Intellectual Property Protection // Proceedings of Custom Integrated Circuits Conference. 2000. P. 517-523.
5. *Gosler James R.* Software protection: Myth or reality? // CRYPTO'85 – Advances in Cryptology. August 1985. P. 140-157.
6. *Aura T. and Gollman D.* Software licence management with smart cards // Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99) USENIX Association. May 1999. P. 75-85,
7. *Cox I.J., Miller M.L., Bloom J.A.* Digital Watermarking. Morgan Kaufmann Publisher // Academic Press. 2002. 542p.
8. *Katzenbeisser S., Petitcolas F.A.P.* Information Hiding: Techniques for steganography and digital watermarking. Artech House. Inc., 2000. 221p.
9. *Wayner P.* Disappearing Cryptography: Information Hiding, Steganography & Watermarking. Second Eddition. Morgan Kaufmann Publisher. Elsevier Science Publisher, 2002. 413p.
10. *Collberg C., Thomborson C. and Low D.* A Taxonomy of Obfucating Transformations,Technical Report N148, Department of Computer Science. The University of Auckland, July, 1997.
11. *Aucsmith D.* Tamper resistant software: an implementation // Lectures Notes in Computer Science. 1996. Vol. 1174. Springer-Verlag. P.317-334.
12. *Wang C., Hill J., Knight J., Davidson J.* Software Tamper Resistance: Obstructing Static Analysis of Program. Technical Report CS-2000-12 University of Virginia, December 2000.
13. *Yarmolik V.N., Nicolaidis M., Kebichi O.* Aliasing-Free Signature Analysis for RAM BIST // Proc. International Test Conference. Washington D.C., October 2-4, 1994. P. 368-377.
14. *Yarmolik V.N.* Calculation of Signatures of Regular Periodic Sequences // Automation and Remote Control. June, 1992. P. 119-127.
15. *Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф.* Защита информации в компьютерных системах и сетях. М.: Радио и связь, 1999. 328с.
16. *Харин Ю.С., Берник В.И., Матвеев Г.В.* Математические основы криптологии. Мн.: БГУ, 1999. 319с.
17. *Yarmolik V.N.* Contents Independent RAM Built In Self Test and Diagnoses based on Symmetric Transparent Algorithm // 3rd Workshop on Design and Diagnostics of Electronic Circuits and Systems DDECS'2000. Smolenice, Slowakia, April 5-7, 2000. P. 220-227.
18. *Yarmolik V.N., Kachan I.V.* Self-Testing VLSI Design // Elsevier Science Publishers. 1993. 345 p.
19. *Wang C., Hill J., Knight J., Davidson J.* Software Tamper Resistance: Obstructing Static Analysis of Program. Technical Report CS-2000-12. University of Virginia. December, 2000.
20. *Torunoglu, Charbon E.* Watermarking-Based Copyright Protection of Sequential Function // IEEE Journal Solid-State Circuits. February, 2000. Vol. 35, N3. P. 434-440.
21. *Monden A., Iida H., Matsumoto K.* A Practical Method for Watermarking Java Programs // The 24th Computer Software and Applications Conference (compsac2000). Taipei, Taiwan. Oct., 2000.
22. *Stern J.P., Hachez G., Koeune F., Quisquater J-J.* Robust Object Watermarking: Application to Code // Lectures Notes in Computer Science (LNCS). A. Pfitzmann, editor. Dresden: Springer-Verlag, 2000. Vol.1768. P. 368-378.
23. *Портянко С.С.* Внедрение водяного знака в ПО на основе использования статистических свойств исполняемого кода // Докл. БГУИР. 2003. Т.1, No2/1. С. 14-15.
24. *Bender W., Gruhl D., Morimoto N., Lu A.* Techniques for data hiding // IBM Syst. J. 1996. Vol. 35.
25. *Захаров В.К., Севастьянов Б.А., Чистяков В.П.* Теория вероятностей. М.: Наука, 1983.
26. Портянко С.С., Ярмолик В.Н.. Защита программных модулей от несанкционированного использования при помощи водяных знаков // Изв. Белорус. инж. акад. 2003. № 1(15)/3. С. 163-165.