ИНФОРМАТИКА

2005 **январь-март** **№ 1**

УДК 681.3.6, 681.324.06

С.С. Портянко, В.Н. Ярмолик

ВНЕДРЕНИЕ ВОДЯНОГО ЗНАКА В ИСПОЛНЯЕМЫЙ КОД НА ОСНОВЕ ПОДСТАНОВОК МАШИННЫХ ИНСТРУКЦИЙ *

Проводится анализ параметров исполняемого кода как объекта внедрения скрытой информации с использованием стеганографических подходов. Представлены результаты исследований, проведенных с целью оценки информационной ёмкости, которая обеспечивается механизмом эквивалентных замен машинных инструкций. Приводится краткое описание новых алгоритмов внедрения водяного знака, основанных на подстановках машинных инструкций.

Введение

Разработчики программных продуктов, получающие доход от их продажи, постоянно сталкиваются с рядом проблем, связанных с нелегальным копированием программного обеспечения. Лица, нарушающие авторские права, могут создать копию продукта для частного использования, осуществления промышленного шпионажа путем обратного проектирования программы и последующей ее продажи как своей собственной с целью получения выгоды [1]. Каждое из перечисленных действий злоумышленника несёт для разработчика потерю части дохода от разработки. Для решения этой проблемы параллельно с развитием технологий создания программного обеспечения (ПО) совершенствуются технологии защиты программ от несанкционированного использования. К ним можно отнести: принятие различных мер по затруднению создания копий программных продуктов, применение технологий дешифрования программного кода «на лету» (во время выполнения), обфусцирование программ – преобразование их исходного кода к виду, максимально затрудняющему понимание алгоритма работы. В последнее время одним из направлений развития средств противодействия компьютерному пиратству стали исследования защиты ПО водяными знаками – средством, позволяющим доказать права собственности на программу целиком или некоторую её часть [2].

Внедрение водяного знака в программный код подразумевает размещение в нем некоторой скрытой информации, для чего применяются стеганографические подходы.

1. Анализ параметров исполняемого кода, обладающих информационной избыточностью

Размещение скрытого сообщения в некотором объекте возможно, если он удовлетворяет следующим требованиям:

– располагает параметром, значение которого имеет природу, максимально близкую к случайной;

– незначительные модификации значений данного параметра не влияют существенно на качество самого объекта.

В случае с графическими изображениями зачастую в качестве такого параметра используют значение яркости точки изображения, которое, как правило, кодируется 8-битным значением. Поэтому в простейшем случае, при внедрении скрытого сообщения в графический образ, модифицируют младшие значащие биты значений яркости точек. При этом модификация младших битов практически не меняет качества изображения.

В случае с программными модулями в качестве искомого параметра можно использовать тип инструкции, находящейся в заданной позиции. Если выбирать позиции инструкций по псевдослучайному закону и вместо символик инструкций рассматривать, например, индекс инструкции в словаре, то индексы выбираемых инструкций будут образовывать после-

* Работа выполнялась при поддержке гранта BMBF BLR 02/006 «DSP-based control systems for safety-related applications».

довательность значений, близких к случайным. Такую последовательность можно рассматривать как аналогичную последовательности значений яркости точек изображения при его построчном сканировании. Для того чтобы удовлетворить второму требованию, необходимо определиться со способом модификации выбранного параметра. Предлагается в качестве механизма модификации использовать замены одних инструкций на другие, эквивалентные, с иным значением индекса. Поскольку рекомендуется осуществлять замены инструкций на функционально эквивалентные, то такая модификация параметра не повлечет за собой нарушение логики работы программы, а лишь незначительно скажется на размере исполняемого кода и скорости работы.

2. Оценка эффективности подстановок инструкций

В случае с графическими изображениями при использовании самого примитивного способа кодирования скрытой информации (путём модификации младшего значащего бита) каждая точка изображения кодирует 1 бит скрываемого сообщения. В отличие от изображений не каждое значение параметра, являющегося индексом инструкции в словаре, может быть модифицировано, т. е. не все инструкции позволяют замену на эквивалентные. Встает вопрос о том, какая информационная емкость может быть достигнута при кодировании скрытой информации в исполняемом коде программы. Для того чтобы оценить значение данной величины, был проведен ряд экспериментов.

Для эксперимента были использованы исходные коды на языке ассемблера для процессоров Intel серии x86 18 программных модулей, среди которых присутствовали графические библиотеки, сетевые, вычислительные и системные модули. Все модули были скомпилированы при помощи компилятора Visual C++. Исходные коды были получены в результате дизассемблирования файлов формата PE (Portable Executable) и DLL (Dynamic Link Library) для IBM PC. Средний размер исследуемых модулей равнялся 300 Кб, минимальный – 40 Кб, максимальный – 564 Кб. Для каждого модуля проводилась оценка следующих величин:

– количества инструкций, которые могут быть использованы для кодирования бинарных последовательностей N_{sym};

– общего количества инструкций, допускающих замену на другие, эквивалентные N_{repl}.

Общее число команд в программе (N_{total}) может быть определено по формуле

$$N_{\text{total}} = N_{\text{repl}} + N_{\text{const}},$$

где N_{const} – количество инструкций, которые не могут быть заменены на эквивалентные. В свою очередь,

$$N_{\text{repl}} = N_{\text{sym}} + N_{\text{dissym}},$$

где N_{dissym} – количество инструкций, допускающих эквивалентную замену, однако только с использованием так называемых «несимметричных» правил подстановки.

Для более наглядного представления полученных результатов (рис. 1) рассматриваются следующие величины:

Sym – процент инструкций, допускающих применение к ним симметричных правил подстановки, определяемый как $N_{\text{sym}} / N_{\text{total}} \cdot 100\%$;

Repl – процент всех заменяемых инструкций, определяемый как $N_{\text{repl}} / N_{\text{total}} \cdot 100\%$.

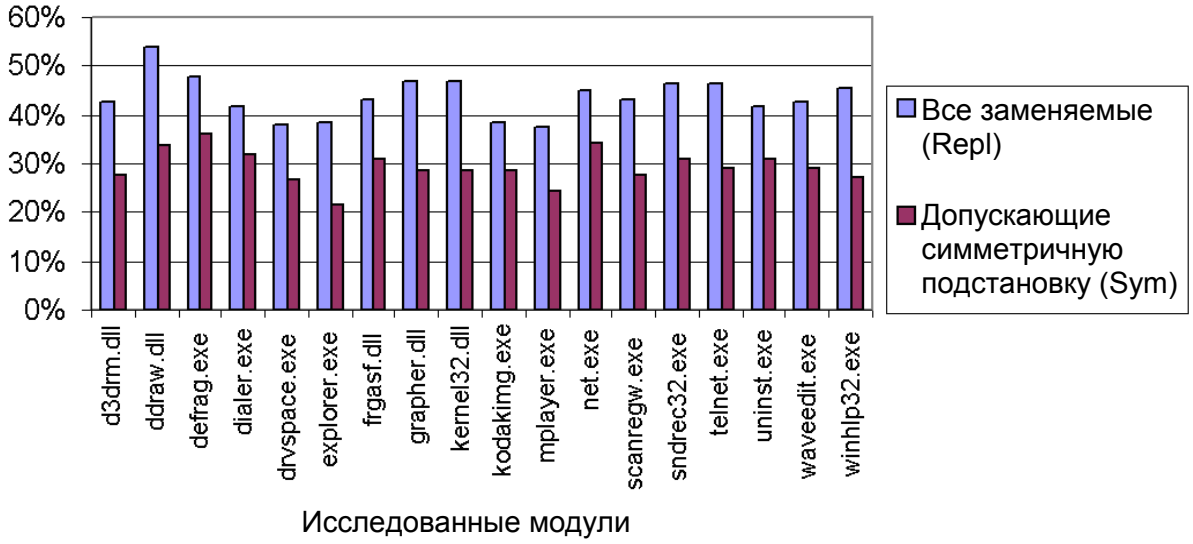

Рис. 1. Значения параметров *Sym* и *Repl*, экспериментально определённые для 18 программных модулей

Для того чтобы осуществить приведенную выше оценку, были использованы два словаря правил подстановок (замен) инструкций. Количество правил в первом словаре равнялось 19, во втором – 29. Каждое правило представляло собой выражение, состоящее из левой и правой частей. В левой части описывались инструкции, подлежащие замене, а в правой – эквивалентная последовательность инструкций. Первый словарь содержал только симметричные правила, в соответствии с которыми возможна замена как левой части выражения на правую, так и правой части на левую. Такие правила подстановки позволяют кодировать двоичные последовательности. Например, с помощью правила

$$(mov\ \%r,0)=(xor\ \%r,\%r)$$

можно закодировать 1 бит информации следующим образом. Если обнуление регистра производится занесением в него нулевого значения (инструкцией *mov*), то кодируется двоичный ноль, а если обнуление происходит посредством выполнения операции «исключающее ИЛИ», то кодируется двоичная единица. Второй словарь включает в себя первый, а также правила, которые не являются симметричными (позволяют замену только левой части на правую) или чья правая часть представляет собой чрезвычайно редко используемую комбинацию инструкций. В качестве примера несимметричного правила можно привести следующее:

$$(or\ \%r,0)=(cmp\ \%r,0).$$

Это правило означает, что инструкция *or*, выполняющая побитовую операцию «ИЛИ» каждого бита регистра с нулевым значением с целью определения факта равенства регистра нулю (флаг *zf*=1), может быть заменена инструкцией *cmp*, которая сравнивает значение регистра с нулем и в случае равенства выставляет флаг *zf* равным единице. Однако обратная замена не всегда является корректной, поскольку инструкция *cmp* может быть использована для определения того, что операнд больше или меньше нуля – в этом случае замена *cmp* на *or* будет некорректной. На диаграмме, изображенной на рис. 2, приведены процентные соотношения инструкций, подлежащих замене.

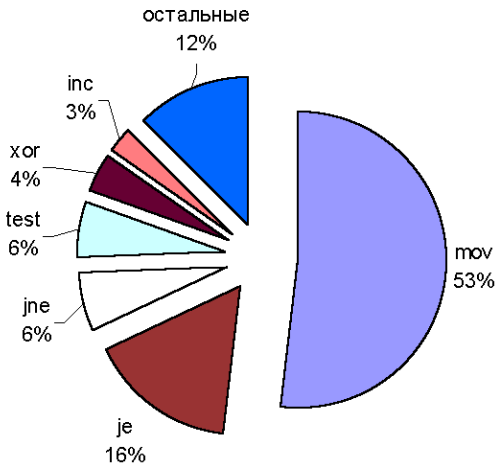

Рис. 2. Соотношение частот встречаемости инструкций

При внедрении скрытой информации путем эквивалентных подстановок возникает проблема изменения частот встречаемости инструкций. Эта проблема становится наиболее очевидной при кодировании двоичных последовательностей с использованием так называемых «симметричных» правил подстановки. Допустим, что в немодифицированной программе инструкция, стоящая в левой части такого правила, имеет частоту использования, которая отличается от частоты использования инструкции, образующей правую часть правила. Далее при помощи «симметричного» правила в коде программы будет закодирована двоичная последовательность. В этом случае соотношение частот использования блоков инструкций из левой и правой частей правила примет значение, близкое к значению соотношения нулей и единиц в закодированной последовательности. Данный факт сделает внедренную информацию уязвимой в случае применения стегоанализа, состоящего в исследовании частот использования инструкций в рассматриваемой программе и сравнении полученных значений со значениями частот, являющимися типичными для программ данного множества.

Для того чтобы избежать описанного выше побочного эффекта, необходимо заботиться о сохранении частот встречаемости инструкций, использованных для внедрения скрытой информации. Возможно применение нескольких подходов к решению этой задачи. Один из них заключается в предварительном преобразовании двоичной последовательности, в результате которого можно было бы получить последовательность с соотношением нулей и единиц, равным заданному. Такое преобразование связано с увеличением длины последовательности и, следовательно, с уменьшением плотности внедряемых данных.

Альтернативный подход к решению описанной задачи заключается в том, что при кодировании двоичной последовательности из рассмотрения исключается некоторое количество экземпляров команды, обладающей большей частотой использования. Такой подход тоже приводит к снижению плотности внедряемых данных. Однако в связи с тем, что при этом подходе часть инструкций (обладающих большей частотой использования) остаётся без изменений, внедрение скрытой информации в меньшей мере сказывается на производительности программы. В проводимой работе была экспериментально исследована эффективность предлагаемого подхода путем оценки обеспечиваемой плотности внедрения скрытой информации.

Рассмотрим пару симметричных инструкций I_1 и I_2, обладающих частотами встречаемости соответственно f_1 и f_2, причём $f_1 > f_2$. Для того чтобы при помощи этих инструкций закодировать двоичную последовательность, соотношение нулей и единиц в которой считается равным единице, и при этом не изменить частоту их встречаемости, при кодировании необходимо использовать одинаковое количество экземпляров инструкций I_1 и I_2. С целью соблюдения этого условия был применен алгоритм, отбрасывающий часть инструкций с большей частотой появления и использующий для кодирования двоичной последовательности примерно одинаковое количество инструкций I_1 и I_2. Разработанный алгоритм отбора инструкций позволяет, сохраняя

оригинальные частоты инструкций, закодировать количество бит, равное количеству экземпляров инструкции, обладающей меньшей частотой появления. Экспериментальная оценка плотности внедряемой информации проводилась при условии использования данного алгоритма.

Оптимальное и максимальное количество бит, которое можно закодировать при помощи использованного алгоритма, определяется следующим образом:

$$Nbits_{\text{opt}} = f_2^{\text{avg}} \cdot N; \quad (1)$$

$$Nbits_{\max} = f_2^{\max} \cdot N, \quad (2)$$

где N – общее количество инструкций в программе; f_2^{avg} – среднее, а $f_2^{\max}$ – максимальное значения частоты использования инструкции I_2, подсчитанные для всего множества исследованных программ.

Полученные результаты (рис. 3) представлены в виде процентных долей всех заменяемых инструкций; инструкций, допускающих симметричную замену; инструкций, которые позволяют замену, но не являются симметричными; оптимальной и максимальной долей инструкций, которые могут быть использованы для кодирования двоичных последовательностей без нарушения частот использования инструкций.

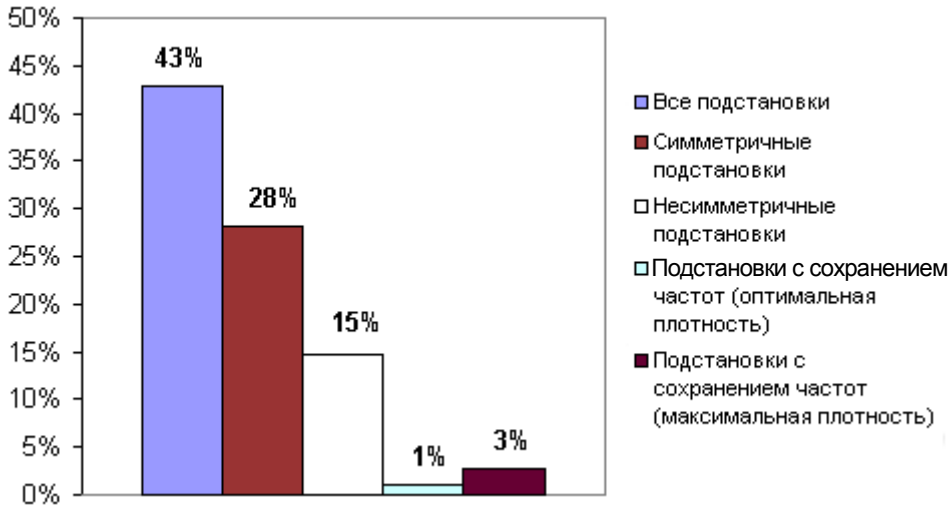

Рис. 3. Результаты экспериментальной оценки исследуемых величин

Алгоритм кодирования, примененный в описанном эксперименте для сохранения частот использования инструкций, не является самым оптимальным, поскольку исключает из рассмотрения не только некоторый процент часто встречаемых инструкций, но и часть редко встречаемых. В случае если для кодирования будет использован оптимальный алгоритм, процент команд, которые могут быть использованы для кодирования без нарушения частот, а значит, и плотность внедрения скрытой информации могут быть увеличены приблизительно в два раза и составить 5% и 1 бит на 1838 инструкций соответственно.

3. Алгоритмы внедрения водяных знаков

Предлагаемые алгоритмы используют статистические свойства исполняемого кода, благодаря чему внедряемые с их помощью водяные знаки гораздо более устойчивы к атакам типа «искажение».

3.1. Подход, основанный на адаптированном алгоритме «Patchwork»

В работе [3] описывается алгоритм «Patchwork», который позволяет поместить водяной знак в растровое изображение и состоит из следующих шагов.

1. С помощью определенного ключа (K), являющегося инициализирующим значением для генератора псевдослучайных чисел, выбирается пара точек изображения (a_i, b_i).

2. Яркость точки a_i увеличивается на некоторую величину (обычно от 1 до 5).

3. Яркость точки b_i уменьшается на ту же самую величину.

4. Шаги 1 – 3 повторяются n раз.

В результате выполнения шагов 1 – 4 изображение наделяется специфической статистической характеристикой, которая проявляется в том, что для выбираемой генератором, проинициализированным ключом K, последовательности точек среднее значение величины $S = a_i - b_i$, будет существенно отличаться от нуля. Эта характеристика не могла возникнуть случайно и может быть обнаружена, только если будет известен ключ K.

Предлагаемый подход базируется на алгоритме «Patchwork». Используется свойство исполняемого кода, состоящее в следующем: если с помощью генератора псевдослучайной последовательности из кода программы выбрать L инструкций, то значение функции $M(L)$ будет отличаться от значения $M(N)$ не более чем на некоторую величину ε, где N – общее количество инструкций в программе. С помощью этого свойства исполняемого кода можно внедрить в него признак авторства путем модификации инструкций, принадлежащих одному из подмножеств L_i (способом эквивалентных замен), таким образом, чтобы

$$\left|M(L_s) - M(N)\right| >> \varepsilon . \qquad (3)$$

В адаптированном варианте «Patchwork», используемом для внедрения водяного знака в исполняемый код, вместо яркостей точек рассматриваются индексы инструкций. В качестве функции M может быть использовано, например, среднее значение индекса инструкции.

3.2. Алгоритм, основанный на автокорреляционных свойствах исполняемого кода

Экспериментально было определено, что значение автокорреляционной функции ($A(k)$) исполняемого кода программы на языке ассемблера для процессоров серии Intel x86 начиная с определённого значения сдвига (k) практически не меняется (рис. 4) и не превышает некоторой величины [4].

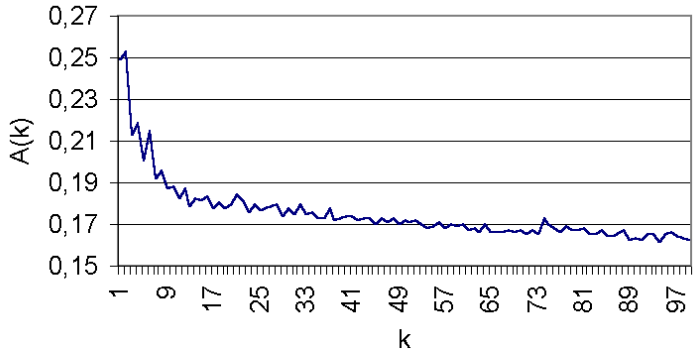

Рис. 4. Экспериментально полученная зависимость значения автокорреляционной функции от величины сдвига

На первом шаге алгоритма инструкции, находящиеся на псевдослучайно выбранных позициях, подвергаются преобразованиям (подстановкам). Последовательность этих преобразований воспроизводится при наличии ключа, который является начальным значением генератора псевдослучайных чисел. На втором шаге оставшееся множество команд модифицируется таким образом, чтобы значение функции $A(k_0)$ для определённого сдвига k_0 было существенно выше значений, принимаемых этой же функцией на других сдвигах k_i. На третьем (последнем) шаге алгоритма воспроизводится последовательность позиций инструкций, модифицированных на первом шаге, и множества применённых к ним преобразований, в результате чего инструкции, модифицированные на шаге 1, принимают первоначальный вид. В результате выполнения

шага 3 алгоритма приобретенное программой свойство (значение $A(k)$) теряется, что позволяет скрыть водяной знак. При этом он может быть обнаружен только при наличии ключа, использованного для выборки инструкций и их подстановок на шагах 1 и 3.

Заключение

Приведенные результаты исследований указывают на то, что в среднем более чем третья часть инструкций программы позволяет осуществлять их подстановку (замену на эквивалентные). При этом наибольший вес среди заменяемых инструкций имеют *mov, je, jne* и *test.* Избыточность, присутствующая в машинном коде, позволяет закодировать в нем двоичные последовательности путём подстановок инструкций, однако плотность внедряемых данных относительно мала по сравнению с возможностями, обеспечиваемыми применением аналогичного подхода к графическим изображениям, аудиофайлам или текстам на естественном языке. Необходимость сохранения частот встречаемости инструкций также приводит к существенному уменьшению информационной емкости. Внедрение в исполняемый код программ водяных знаков, являющихся признаками авторства, путем внесения изменений на уровне значений статистических характеристик [5] не требует использования только «симметричных» правил подстановки, а также допускает применение механизма перестановок инструкций. Данные особенности делают этот подход наиболее приемлемым.

Список литературы

1. Experience with Software Watermarking / J. Palsberg, S. Krishnaswamy, M. Kwon et al. // Proc. 16th Annual computer security applications conference (ACSAC '00). – New Orleans, Louisiana, December 11–15, 2000. – P. 308–316.

2. Collberg C., Thomborson C. Watermarking, tamper proofing and obfuscation – tools for software protection. – University of Auckland, 2000.

3. Bender W., Morimoto G.N., Lu A. Techniques for data hiding // IBM Syst. J. – V. 35. – P. 313–336.

4. Портянко С.С., Ярмолик В.Н. Защита программных модулей от несанкционированного использования при помощи водяных знаков // Известия Белорусской инженерной академии. – 2003. – № 1(15)/3. – С. 163–165.

5. Ярмолик В.Н., Портянко С.С. Статистические свойства исполняемого кода приложений и их использование для защиты авторского права // Вести Института современных знаний. – № 1(18). – 2004. – С. 62–69.

Поступила 11.06.04

Белорусский государственный университет информатики и радиоэлектроники,
Минск, П. Бровки, 6
e-mail: partsianka@bsuir.unibel.by
yarmolik@bsuir.unibel.by

S.S. Partsianka, V.N. Yarmolik

EXECUTABLE CODE WATERMARKING BASED ON SUBSTITUTION OF MACHINE INSTRUCTIONS

In the first part of paper the analysis of executable code parameters as an object of hidden information embedding by means of steganographic approaches is performed. The second part contains the results of studies to estimate the information capacity provided by equivalent substitutions of machine instructions. In the third part the brief description of new watermarking algorithms based on instructions substitution is given.