Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

УДК 681.3.06:620.9.002.5

ПЕТРИК
Сергей Юрьевич

ЗАЩИТА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ОТ ОБРАТНОГО ПРОЕКТИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ЗАПУТЫВАЮЩЕГО ПРЕОБРАЗОВАНИЯ

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук
по специальности 05.13.19 – Методы и системы защиты информации, информационная безопасность

Минск, 2008

Работа выполнена в учреждении образования «Белорусский государственный университет информатики и радиоэлектроники»

Научный руководитель	Ярмолик Вячеслав Николаевич, д-р техн. наук, профессор, профессор кафедры программного обеспечения информационных технологий учреждения образования «Белорусский государственный университет информатики и радиоэлектроники»
Официальные оппоненты:	Голенков Владимир Васильевич, д-р техн. наук, профессор, заведующий кафедрой интеллектуальных информационных технологий учреждения образования «Белорусский государственный университет информатики и радиоэлектроники»,
	Захаров Владимир Владимирович, канд. техн. наук, УП «Творческая лаборатория»
Оппонирующая организация	Белорусский государственный университет

Защита состоится 26 июня 2008г. в 14.00 на заседании совета по защите диссертаций Д 02.15.06 при учреждении образования «Белорусский государственный университет информатики и радиоэлектроники» по адресу: 220013, г. Минск, ул. П. Бровки, 6, БГУИР, корп. 1, ауд. 232, тел. 293-89-89.

С диссертацией можно ознакомиться в библиотеке учреждения

КРАТКОЕ ВВЕДЕНИЕ

С развитием информационных технологий и распространением программного обеспечения (ПО) обостряются проблемы его несанкционированного использования, поэтому одним из актуальных вопросов для производителей ПО является его защита от незаконного копирования, присвоения алгоритмов или их частей для использования в конкурирующих продуктах, несанкционированной модификации, которая может преследовать различные злонамеренные цели.

Для защиты ПО от перечисленных атак применяются меры, которые принято разделять на юридические и технические (последние делятся на аппаратные и программные). Существующие методы защиты обладают хорошо изученными достоинствами и недостатками, однако распространение таких платформ, как .NET и Java, программы для которых достаточно просто декомпилируются, показало необходимость разработки новых методов. Одним из новых направлений, решающих задачу защиты таких программ, является запутывающее преобразование (obfuscation), задача которого – максимальное усложнение процесса обратного проектирования.

В публикациях по теме запутывающего преобразования рассматриваются аспекты его теоретического и практического применения, однако многие вопросы остаются открытыми. Например, проблемой является то, что запутывающее преобразование, как считают многие авторы, не способно защитить отдельные модули от несанкционированного использования целиком как «черный ящик» без уяснения логики их работы.

Другая проблема – использование на практике преобразований без исследования вопроса их эффективности. Поэтому актуальной является разработка новых методов запутывания, которые были бы основаны на исследовании свойств ПО, и методики оценки эффективности запутывающего преобразования.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Связь работы с крупными научными программами (проектами) и темами

Исследования проводились в рамках задания ГБЦ 02-3086 «Разработать теоретические основы построения систем защиты цифровой интеллектуальной собственности в современных условиях развития информационных технологий» Государственной программы ориентированных фундаментальных исследований «Научные основы новых информационных технологий» («Инфотех»), выполненной НИЛ 3.3 БГУИР; в рамках задания Т06-М 141 «Разработка стеганографических методов защиты авторских прав на программное обеспечение» Белорусского Республиканского фонда фундаментальных исследований (ГБЦ 06-7016); в рамках международного

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения, списка литературы и двух приложений. В первой главе проводится аналитический обзор литературы по теме защиты программного обеспечения в целом и запутывающего преобразования в частности, выделяются открытые проблемы, подлежащие решению в рамках диссертации. Во второй главе разрабатывается модель оценки эффективности запутывающего преобразования на основе метрик. В третьей главе приводятся результаты исследования потенциала различных методов запутывания, разрабатываются алгоритмы новых методов. Четвертая глава включает описание разработанного программного комплекса «Obfuscation Studio» и результаты его практического использования. В заключении приводятся выводы о научной и практической значимости работы.

Диссертация изложена на 128 страницах машинописного текста, в том числе основная часть – на 103 страницах, содержит 46 рисунков, 15 таблиц, состоит из введения, четырёх глав, заключения, списка литературы, включающего 102 названия, и 2 приложений.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во **введении** обоснована актуальность темы работы.

В **первой главе** проведен анализ возможных атак на программное обеспечение, основные из них выделены согласно положениям закона Республики Беларусь об авторском праве и смежных правах:

– копирование программы с другими целями, нежели разрешенными пунктом 1 статьи 21 закона об авторском праве (в англоязычной литературе обозначается термином pigacy);

– присвоение алгоритмов или их частей для использования в конкурирующих продуктах (reverse engineering);

– несанкционированное изменение, которое может преследовать такие цели, как отключение защитных механизмов, удаление цифровых подписей, водяных знаков и прочие, противоречащие пункту 2 статьи 21 (tampering).

Выявлено, что в реализации двух последних атак значительное место занимают приемы обратного проектирования, такие как:

– дизассемблирование – извлечение из программы инструкций на ассемблерном языке;

– декомпиляция – процесс извлечения из машинного или промежуточного представления программы ее представления на языке высокого уровня.

Анализ литературы по теме защиты ПО позволил сделать вывод, что наиболее уязвимыми для обратного проектирования, исследования, модификации и воровства алгоритмов и их частей являются программы, поставляемые в виде исходных кодов (скрипты), и программы, написанные для платформ, использующих компиляцию не в машинные инструкции, а в сборки

на промежуточных языках (MSIL для платформы .NET и Java-bytecode для платформы Java).

Определены основные методы противодействия атакам на программное обеспечение: правовые и технические (аппаратные и программные). Показано, что большинство этих подходов обладают недостатками: аппаратные средства защиты не всегда удобно распространять, они поддаются эмуляции и т.д., правовые декларируют ответственность сторон, участвующих в разработке и распространении ПО, но не гарантируют выполнение таких соглашений. Выявлена ограниченность для использования программных методов защиты: использования серийных номеров и шифрования.

Одним из новых методов защиты ПО от обратного проектирования, а значит, и исследования, модификации, использования алгоритмов является запутывающее преобразование. Первые публикации по данной тематике появились в 1997 году. Было дано следующее определение: *запутывающее преобразование* – это трансформация программы в функционально эквивалентную исходной, однако менее подверженную приемам обратного проектирования. Были выделены перспективные методы запутывающего преобразования:

– лексические преобразования – замена имен идентификаторов и изменение внешнего вида исходных текстов программ (для скриптов) за счет их переформатирования;

– запутывание структур данных – изменение типов данных, разделение и слияние отдельных переменных, преобразование массивов и т.п.;

– изменение потока выполнения программ – преобразование программ таким образом, что они насыщаются нелинейными конструкциями: ветвлениями и циклами.

Наука о запутывающем преобразовании достаточно нова, что обусловило отсутствие у нее формального и математического фундамента, и представляет собой совокупность разрозненных теоретических работ и практических реализаций запутывающих преобразователей. Выявлено, что эти запутывающие преобразователи реализуют широкий спектр приемов запутывания, однако недостаточное теоретическое изучение и практическое тестирование таких приемов часто приводит к невозможности нормального функционирования защищаемых программ. Проблемой является и то, что запутывающее преобразование, как считают многие авторы, не способно защитить отдельные модули от несанкционированного использования целиком как «черный ящик» без уяснения логики их работы.

Анализ литературы показал, что оценить эффективность запутывающего преобразования можно двумя методами: при помощи формальной модели программы, которая на сегодняшний день не разработана, и при помощи метрик сложности и производительности программного обеспечения. Однако цельной концепции оценки при помощи метрик не выявлено, что делает ее разработку одной из нерешенных задач запутывающего преобразования.

В результате проведенного анализа были поставлены цель и задачи, подлежащие решению в рамках диссертации, которые можно обобщить

следующим образом: создание новых приемов запутывающего преобразования с оцененной эффективностью и формальными моделями представления алгоритмов запутывания.

Во **второй главе** представлена модель оценки эффективности запутывающего преобразования, основанная на метриках, которые обладают описанными в диссертации свойствами.

Отмечено, что при запутывании программ возрастают не только значения метрик сложности (что и требуется для запутывания), но и потребляемые ресурсы: объем памяти и время выполнения программы. Принимая во внимание этот факт, программу можно представить в виде вектора в *n*-мерном пространстве измеренных для нее метрик сложности и потребляемых ресурсов. Формально это пространство изображается следующим образом:

$$P = R^{(n)} = \{(x_1, x_2 ..., x_n) | \, x_i \in R, i \in [1, n]\}, \qquad (1)$$

где P – пространство, в котором могут находиться представления программ $\vec{p}_1, \vec{p}_2 ... \vec{p}_k$ (k – количество представлений программ);

$R^{(n)}$ – n-мерное пространство;

n – количество метрик, которые были определены для данной программы;

R – множество рациональных чисел;

x_i – значение некоторой метрики, которая была измерена для данной программы.

Показано, что для данного пространства выполняются все аксиомы, необходимые для того, чтобы считать пространство линейным, из чего следует выполнение для него различных утверждений и математических соотношений, одно из которых – определение длины вектора в линейном пространстве.

Отмечено, что изменение значений различных метрик не одинаково влияет на сложность программы. Этот факт вызвал необходимость использования матрицы весовых коэффициентов различных метрик для того, чтобы при умножении на нее вектора представления программы нормализовать и ранжировать их и затем оперировать их абсолютными значениями.

Матрица весовых коэффициентов представляет собой диагональную матрицу, которая имеет ту же размерность, что и пространство программы, и в своих диагональных элементах содержит неотрицательные числа, отражающие коэффициенты нормализации и экспертные представления о влиянии на сложность программы изменения различных метрик.

Произведением матрицы весовых коэффициентов на вектор представления программы будет вектор, определяемый следующим образом:

$$\vec{p}' = Y \cdot \vec{p} = (y_1 \cdot x_1, y_2 \cdot x_2 ..., y_n \cdot x_n), \qquad (2)$$

где Y – матрица весовых кэффициентов;

$y_1, y_1 \ldots y_n$ – диагональные элементы матрицы Y.

Кроме метрик, которые необходимо максимизировать, есть и такие, которые должны быть минимизированы. К ним относятся длина программы и время ее выполнения. Поэтому вектор представления программы рассматривается как сумма двух векторов:

$$\vec{p}' = \vec{E}\uparrow + \vec{E}\downarrow,$$
$$E\uparrow = \left|\vec{E}\uparrow\right| = \sqrt{(y_1x_1)^2 + (y_2x_2)^2 + \ldots + (y_{n-2}x_{n-2})^2}, \quad (3)$$
$$E\downarrow = \left|\vec{E}\downarrow\right| = \sqrt{(y_{n-1}x_{n-1})^2 + (y_nx_n)^2},$$

где $E\uparrow$ – длина вектора, которую необходимо максимизировать,

$E\downarrow$ – длина вектора, которую необходимо минимизировать,

$x_1, x_2, \ldots x_{n-2}$ – значения метрик сложности,

x_{n-1}, x_n – длина программы и время ее выполнения.

Очевидно, что наилучшим является запутывающий преобразователь, который оптимизирует код по времени и (или) по объему памяти (уменьшает значение модуля вектора $E\downarrow$) и вместе с тем делает его сложнее (увеличивается модуль вектора $E\uparrow$).

Графическое представление модели показано на рисунке 1:

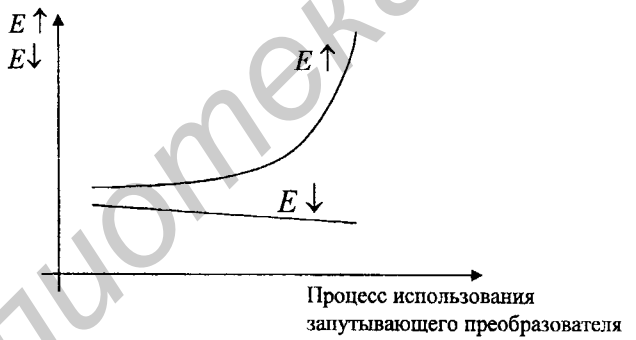


Рисунок 1 – Изменение модуля вектора представления программы в процессе запутывающего преобразования

Графики $E\uparrow$ и $E\downarrow$, изображенные на рисунке 1, условны, их цель – показать, что идеальный запутывающий преобразователь такой, что при постоянстве или уменьшении длины кода и времени его выполнения экспоненциально увеличивается сложность программы.

Таким образом, предложено представление программы как вектора в линейном пространстве, позволяющее оценить эффективность запутывающего преобразования и сравнить различные методики запутывания, анализируя изменения длин векторов сложности и потребляемых ресурсов.

Кроме того, в главе 2 приведены практические указания по использованию предложенной модели, позволяющие в автоматическом режиме определять эффективность запутывающего преобразования по полученным значениям метрик.

В заключении главы приводится представление свойств запутывающего преобразования (эффективность, функциональность, запутанность) с точки зрения разработанной модели.

В **третьей главе** проведено статистическое исследование существующих методов запутывающего преобразования: лексического и внедрения в код селектирующих выражений, и представлены алгоритмы новых методов: искусственного увеличения сцепления модулей и изменения структуры данных программы.

Для *лексического метода* запутывания построена модель, позволяющая оценить его эффективность как долю программы, которую составляют идентификаторы. Исследование показало, что вне зависимости от способа подсчета доля идентификаторов колеблется в среднем в пределах от 15 до 55 % для всех рассмотренных языков: C, C++, C#, Java.

Кроме того, был исследован вопрос распределения длин идентификаторов, что позволило выявить, какими они должны быть, чтобы факт запутывания невозможно было определить простым подсчетом их длин. Установлено, что средняя длина идентификаторов составляет 7–8 символов.

Представлена формальная модель запутывания лексическим методом (4):

$$\{V, W\} \xrightarrow{T\_лексич.} \{V', W\}, |V'| \le |V|, v_{l...k} \to v'_j, \qquad (4)$$

где V – множество идентификаторов, объявленных в этой программе;

W – множество всех остальных слов, кроме комментариев (ключевые слова языка, имена структур данных и методов, определенных в подключаемых библиотеках, числа, отдельно взятые строки и т. п.);

V' – множество идентификаторов после выполнения процедуры переименования;

v, v' – элементы множеств V, V';

l, k, j – индексы элементов множеств V, V'.

Следующий метод запутывающего преобразования, рассмотренный в главе, – *метод внесения в код селектирующих выражений*. Для оценки эффективности такого метода предложены следующие метрики:

M1 – насыщенность программы селектирующими операторами;

M2 – средняя глубина вложенности операторов ветвления;

M3 – время выполнения программы (в миллисекундах);

M4 – длина программы (в байтах).

Определено, что среднее значение метрики M1 (количество операторов, используемых в селектирующих выражениях по отношению к общему количеству слов программы) для программ на языках C, C++, C#, Java составляет 0,055 (5,5 %). Среднее значение метрики M2 составляет около 1,75.

Представлена формальная модель запутывания программ этим методом.

Далее в главе представлен авторский *метод искусственного повышения сцепления модулей*. Для оценки эффективности данного метода используется одна из распространенных метрик сложности программного обеспечения – CBO (сцепление между объектами, Coupling Between Objects), показано, что она полностью удовлетворяет свойствам, приведенным в главе 2, и достаточна для оценки такого фундаментального свойства, как сцепление модулей.

Влияние классов друг на друга осуществляется посредством использования в коде одного класса полей, свойств, методов и событий другого класса. Взаимодействие может происходить непосредственно с классом в том случае, если указанные составляющие класса являются статическими. В противном случае взаимодействие происходит через объект класса.

Класс представляет собой совокупность своих составляющих (полей, свойств, методов, событий), которые и производят взаимодействие. Исходя из этого утверждения, модель взаимодействия классов можно представить следующим образом: пусть A_{ij} – j-й составляющий элемент i-го класса (C_i) программы, тогда его взаимодействие с другими классами выражается параметром $INT(A_{ij})$, определяемым следующим образом:

$$INT\left(A_{ij}\right)=\left\{A_{kl}\,\middle|\,k=\overline{1,n},k\neq i,l=\overline{1,m_k},A_{ij}\Leftrightarrow A_{kl}\right\}, \tag{5}$$

где k – номер класса в программе,

l – номер составляющего элемента в k-м классе,

n – количество классов в программе,

m_k – количество составляющих элементов в k-м классе,

$\Leftrightarrow$ – в данном контексте означает взаимодействие.

Абстрагируясь от того, с каким именно элементом класса взаимодействует элемент A_{ij}, получим множество классов, с которыми он взаимодействует:

$$INT\left(A_{ij}\right)=\left\{C_k\,\middle|\,k=\overline{1,n},k\neq i,A_{ij}\Leftrightarrow C_k\right\}. \tag{6}$$

Тогда взаимодействие класса C_i с другими классами программы можно представить следующим образом:

$$INT\left(C_i\right)=\bigcup_{j=1}^{m_i}INT\left(A_{ij}\right), \tag{7}$$

где $INT(C_i)$ – множество классов, с которыми взаимодействует исследуемый класс,

m_i – количество составляющих элементов в i-ом классе.

Метрику CBO можно выразить следующей формулой:

$$CBO(C_i) = |INT(C_i)|. \quad (8)$$

С другой стороны, составляющие элементы класса могут состоять из нескольких выражений:

$$A_{ij} = \{a_k \mid k = \overline{1,l}\}, \quad (9)$$

где a_k – k-е выражение, составляющее A_{ij},

l – количество выражений в A_{ij}.

Данное представление можно применить для методов классов (в терминах объектно-ориентированного программирования).

Для создания связи между классами необходимо, чтобы код одного класса использовал код другого. В диссертации предложен метод искусственного внедрения связей между классами за счет перемешивания инструкций методов.

Приведенный выше метод связывания классов можно описать следующим образом:

$$\begin{aligned} M_{i_1 j_1} &= M_{i_1 j_1} + part(M_{i_2 j_2}) = \\ &= \{a_{k_1} \mid k_1 = \overline{1,l}\} \cup \{a_{k_2} \mid k_2 \in \overline{l_1, l_2}, l_1 \in [1,m], l_2 \in [l_1, m]\}, \end{aligned} \quad (10)$$

где $M_{i_1 j_1}$ – j_1-й метод класса i_1,

part(M) – часть метода M,

a_{k_1} – оператор метода $M_{i_1 j_1}$

l – количество операторов в методе $M_{i_1 j_1}$,

a_{k_2} – оператор метода $M_{i_2 j_2}$,

m – количество операторов в методе $M_{i_2 j_2}$.

Выражение (10) является основой запутывающего преобразования методом увеличения сцепления модулей.

Последним методом, который исследуется в главе, является *метод изменения структур данных*. Определено, что обращения к переменным составляют в среднем от 10 до 40 % слов модулей программ. Соответственно, применяя различные приемы изменения структуры данных, получим запутывание большой (и достаточно равномерно распределенной по программе) части исходного кода. Это говорит о том, что идея изменения структуры данных может оказаться весьма плодотворной. Кроме такой метрики, как доля программы, которая запутывается методом изменения данных, вводятся следующие метрики оценки эффективности данного метода:

– DOC – количество операций с элементами данных (Data Operations Count);

– LOC – недостаток связности (Lack of Cohesion), показывающий насколько сильно взаимодействие различных частей одного модуля через

объявленные в нем поля. Исследованы значения LOC для тестовых приложений. Значения LOC находятся в пределах от 15 до 100, что говорит о большой зависимости этого значения от специфики программ.

Кроме того, в работе представлена методика, позволяющая определить подмножество переменных, которые используются чаще других. Автоматическое выделение такого подмножества может быть полезным для стратегий запутывания данных, использующих факт существования такого подмножества.

В работе предлагается реализовать следующий алгоритм запутывания данных:

1) вынести все локальные переменные в поля классов, что увеличит время их жизни;

2) преобразовать все поля классов к единому базовому типу;

3) поместить все поля классов в общий для класса массив элементов базового типа.

Для данного алгоритма запутывания характерно следующее:

– реальный тип переменных не известен злоумышленнику, дальнейшее развитие этой особенности алгоритма состоит во включении в код селектирующих выражений, в разных ветвях которых выполняется приведение элементов массива к разным типам;

– в массив могут быть помещены дополнительные запутывающие элементы, которые будут изменяться в ходе программы, заставляя следить за ними взломщика, но не влияющие на алгоритм до того момента, когда в ту же ячейку памяти не будет записано реально использующееся значение;

– массив может перемешиваться, меняя местами значащие и незначащие элементы и др.

Далее в главе рассматривается возможность использования различных методов запутывающего преобразования по отношению к элементам программ, написанных для платформы .NET (классы, структуры, интерфейсы, перечисления).

Делается вывод о возможности реализации каждого из рассмотренных методов согласно предложенным алгоритмам.

В **четвертой главе** рассматривается разработанное приложение Obfuscation Studio и отдельные модули, реализующие методы лексического запутывающего преобразования, внедрения в код предикатов, искусственного повышения сцепления модулей и изменения структуры данных программ. Схема Obfuscation Studio приведена на рисунке 2. За счет использования технологий позднего связывания и отражения достигается способность программы к расширяемости в отличие от иных программ того же назначения.

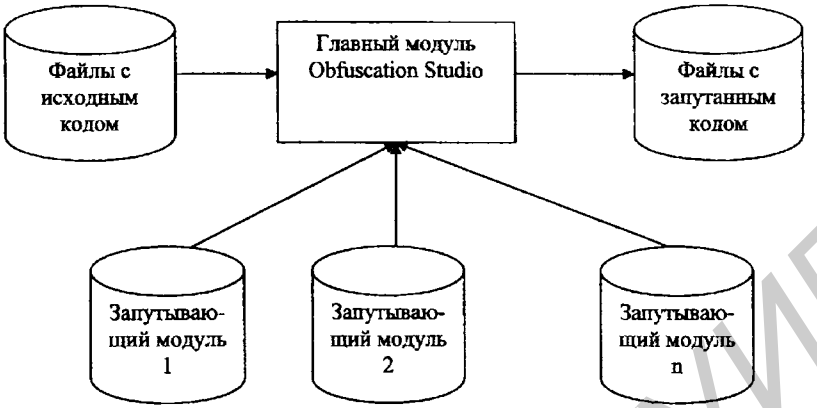


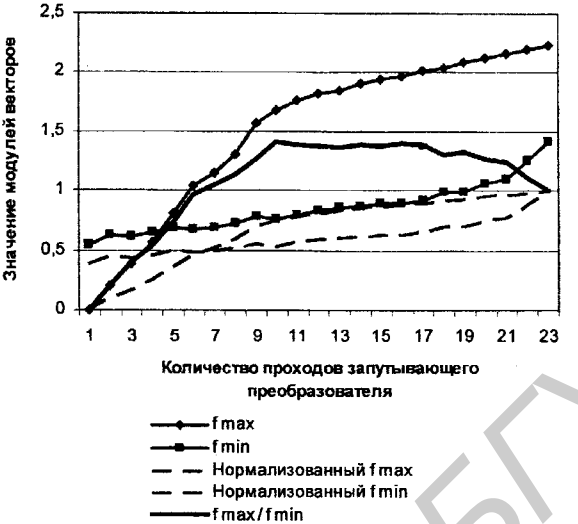
Рисунок 2 – Схема программы Obfuscation Studio

В главе представлена схема реализации запутывающего преобразования *лексическим методом*. Отличительными особенностями данной реализации являются:

– процедура вынесения локальных переменных в поля классов, что позволяет сохранить их запутанные имена при компиляции (так как имена локальных переменных при компиляции, как правило, теряются, что не позволяет их запутать);

– возможность работы в нескольких режимах переименования идентификаторов: из схожих по начертанию символов (l и 1, O и 0), что в значительной степени затрудняет визуальное различение идентификаторов, или из идентичных по написанию букв английского и русского алфавитов (таких как «а», «е», «о», «р»).

Далее в главе представлена относительно простая *реализация метода внедрения в код селектирующих выражений*, которая основана на многократном копировании одних и тех же участков кода с помещением ветвей под логические выражения, всегда принимающие значение «истина» или «ложь». С помощью модели, предложенной в главе 2, было проведено исследование эффективности данного запутывающего преобразования для разного числа повторений процесса запутывания одного и того же участка кода. Результат исследования приведен на рисунке 3.



f_{max} – функция изменения длины вектора $E\uparrow$, f_{min} – функция изменения длины вектора $E\downarrow$

Рисунок 3 – Определение наилучшей стратегии запутывания

Определено, что для данной реализации метода затраты на его использование растут быстрее получаемого полезного эффекта: f_{min} растет быстрее f_{max}. Кроме того, видно, что наилучшие результаты эта реализация для конкретной рассматриваемой программы показывает при 10 проходах запутывающего преобразователя (именно при этом значении количества запутывающих проходов отношение $E\uparrow$ к $E\downarrow$ максимально).

Далее приводится алгоритм работы модуля запутывающего преобразования, реализующего *искусственное увеличение сцепления* модулей программы. Общая схема алгоритма следующая:

1. Анализ кода, который отвечает на следующие вопросы:
   - из каких модулей состоит программа;
   - из каких подмодулей (методов) состоит каждый модуль;
   - использует ли каждый из подмодулей другие модули;
   - каково значение метрики СВО для каждого из модулей.
2. Фаза запутывающего преобразования, которая преследует следующие цели:
   - увеличить значения метрики СВО для каждого модуля;
   - увеличить количество связывающих пар (точек соединения двух модулей), что приведет к увеличению сложности понимания программы.
3. Повторный анализ кода, который определяет, каким образом изменилось значение метрики СВО для каждого из классов, и считает количество связывающих пар между каждым из классов.

Данный алгоритм выполнен слабо детерминированным, с большим влиянием генератора псевдослучайных чисел. Это сделано для того, чтобы

максимально запутать злоумышленника, не позволяя ему выявить строгую закономерность внедрения одних модулей в другие.

Был проведен эксперимент использования данного метода над приложением, состоящим из 17 классов. Результат использования приведен на рисунке 4.

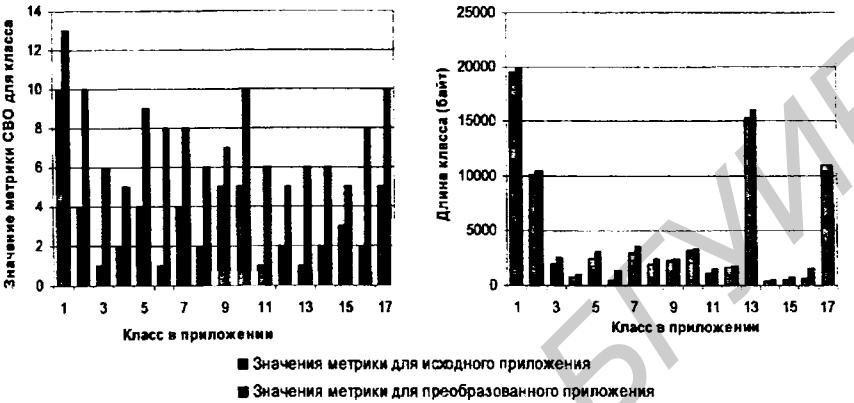


Рисунок 4 – Изменение метрики CBO и длины программы для исходного и преобразованного приложений

Исследование показало, что среднее значение метрики CBO для классов, составляющих тестовую программу, увеличилось в 2,4 раза, в то время как суммарная длина всех классов возросла в 1,10 раз, время выполнения программы возросло не более чем на 1 %. Это говорит о том, что для данного приложения и для данной реализации алгоритма запутывания функция увеличения сложности понимания программы растет в 2,15 раз быстрее функции увеличения объема кода.

Следующим рассмотренным методом является *метод изменения структуры данных* программы. Реализован алгоритм, предложенный в главе 3, и проведено исследование тестовых программ. Оно показало, что структура данных значительно модифицируется, что может быть отражено изменением значений метрики LOC. Результаты измерений этой метрики до и после запутывания приведены в таблице.

Таблица – Изменение значений метрики LOC

Приложение	Значение метрики LOC до запутывания	Значение метрики LOC после запутывания
Obfuscation Studio	102,4286	0,02743
Reports Generator	17,68421	3,78947
Mono.Cecil	98,61842	0,44737
Mono.ILAsm	15,71739	0,07609
Mono.ZipLib	77,21951	0,04878
SharpDevelop	74,57506	8,22564

Многократное изменение значения метрики LOC говорит о том, что модули становятся более целостными, единое поле используется во всех методах классов, что заставляет злоумышленника следить за ним не в некоторых элементах модуля, а во всем модуле.

Подробно был исследован небольшой модуль, реализующий математические преобразования. Определено, что длина вектора сложности программы возросла в 1,1 раза больше, чем длина вектора потребляемых ресурсов. Небольшая фактическая эффективность данного метода компенсируется возможностью использования проработанных приемов мутации массивов «на лету», так как все переменные после применения разработанного метода представляют собой единый массив элементов.

Программный комплекс «Obfuscation Studio» реализован и практически используется для защиты программ, написанных для платформы .NET, от обратного проектирования, незаконного исследования алгоритмов и модификации.

ЗАКЛЮЧЕНИЕ

Основные научные результаты диссертации

1. Предложена модель оценки эффективности запутывающего преобразования, основанная на представлении программы в виде вектора в пространстве измеренных для нее метрик сложности и потребляемых ресурсов, и отличающаяся возможностью использования для сравнения различных подходов к запутыванию [5–А, 6–А, 10–А, 11–А, 12–А].

2. Представлены оценки эффективности существующих методов запутывающего преобразования: лексического метода (в среднем сложность программы возрастает на 35%) и метода внедрения в код селектирующих выражений (возрастание сложности программы в 1,5 раза превышает возрастание потребляемых ресурсов), что позволяет сделать вывод о возможности их использования на практике [1–А, 3–А, 7–А, 8–А, 9–А, 14–А].

3. Разработан метод искусственного увеличения сцепления модулей при помощи внедрения инструкций одних модулей в другие, который, в отличие от существующих методов, противодействует такой атаке как использование модуля целиком без исследования алгоритмов его работы (возрастание сложности программы в 2,2 раза превышает возрастание потребляемых ресурсов) [6–А]; впервые представлен алгоритм изменения структуры данных программ, основанный на преобразовании максимального числа переменных в единый массив базового типа (возрастание сложности программы в 1,1 раза превышает возрастание потребляемых ресурсов), позволяющий использовать различные приемы запутывания массивов; представлена методика определения подмножества элементов данных, которые используется чаще других, которая может быть использована в дальнейших исследованиях запутывания структуры данных [13–А].

4. Разработан программный комплекс Obfuscation Studio, преимущества которого заключаются в расширяемости и простоте использования, реализующий исследуемые существующие и авторские методы запутывающего преобразования [2–А, 3–А, 4–А].

Рекомендации по практическому использованию результатов

Модель оценки эффективности и результаты исследования эффективности различных методов, могут использоваться как исходные данные для дальнейших исследований и при разработке соответствующего программного обеспечения.

Новые методы запутывающего преобразования могут быть использованы как модули, реализованные в рамках Obfuscation Studio, для защиты программного обеспечения от обратного проектирования.

СПИСОК ПУБЛИКАЦИЙ СОИСКАТЕЛЯ ПО ТЕМЕ ДИССЕРТАЦИИ

Статьи в журналах

1–А. Петрик, С.Ю. Исследование лексического метода запутывания исходных текстов программ с целью их защиты / С.Ю. Петрик, В.Н. Ярмолик // Информатика. – 2004. – №3. – С. 58–66.

2–А. Петрик, С.Ю. Использование обфускации для защиты интеллектуальной собственности / С.Ю. Петрик, В.Н. Ярмолик // Известия Белорусской инженерной академии. – 2004. – №1(17)/2. – С. 168–171.

3–А. Петрик, С.Ю. Запутывающее кодирование / С.Ю. Петрик // Управление защитой информации. – 2005. – № 2(9). – С. 182–186.

4–А. Petryk, S. Obfuscation studio executive / S. Petryk, V. Yarmolik // World Academy of Science, Engineering and Technology. – 2005. – Vol. 10. – P. 7–11.

5–А. Петрик, С.Ю. Математическая модель оценки эффективности запутывающего кодирования / С.Ю. Петрик // Инженерный вестник. – 2006. – №1(21)/1. – С. 99–102.

6–А. Петрик, С.Ю. Запутывающее кодирование методом изменения сцепления модулей / С.Ю. Петрик, В.Н. Ярмолик // Автоматика и вычислительная техника. – 2006. – №2. – С. 39–46.

Материалы конференций и тезисы докладов

7–А. Петрик, С.Ю. Исследование эффективности обфускации исходных текстов про-грамм / С.Ю. Петрик, В.Н. Ярмолик // Технические средства защиты информации: тезисы докладов II-ой Белорусско-российской науч.-техн. конф., Минск–Нарочь, 17-21 мая 2004 г. – Минск, 2004. – С. 27.

8–А. Петрик, С.Ю. Запутывающее кодирование методом трансформации потока выполнения программы внесением в код селектирующих выражений / С.Ю. Петрик // Новые информационные технологии в научных исследованиях и в образовании (НИТ-2005): тезисы докладов X всероссийской науч.-техн. конф. студентов, молодых ученых и специалистов, Рязань, 10–12 апр. 2005 г. – Рязань, 2005. – С. 82–84.

9–А. Petryk, S. Efficiency of obfuscation method based on control transformation / S. Petryk, I. Mrozek, V. Yarmolik // Advanced Computer Systems (ACS'2006): proceedings of XIII International Multi-Conference, Miedzyzdroje, Poland, 23–26 oct. 2006. – Vol. 1. – P. 351–359.

10–А. Петрик, С.Ю. Модель оценки эффективности запутывающего кодирования / С.Ю. Петрик // Информационные технологии и кибернетика: сб. докладов и тезисов IV Междунар. науч.-практ. форума, Днепропетровск, 27–28 апр. 2006 г., – Днепропетровск, 2006. – С. 58–59.

11–А. Петрик, С.Ю. Оценка эффективности использования запутывающего кодирования для защиты программного обеспечения / С.Ю. Петрик // Новые математические методы и компьютерные технологии в проектировании,

производстве и научных исследованиях: материалы IX Республиканской науч. конф. студентов и аспирантов, Гомель, 13–15 марта 2006 г., Гомель, 2006. – С. 279–280.

12–А. Петрик, С.Ю. К вопросу определения термина «запутывающее кодирование» («обфускация») / С.Ю. Петрик // Единое информационное пространство: сб. докладов IV Междунар. науч.-практ. конф., Днепропетровск, 7–8 декабря 2006 г., – Днепропетровск, 2006. – С. 71.

13–А. Петрик, С.Ю. Запутывающее преобразование путем изменения структуры данных программ / С.Ю. Петрик // Технические средства защиты информации: материалы докладов и краткие сообщения V Белорусско-российской науч.-техн. конф., Минск, 28 мая–1 июня 2007 г., Минск, 2007. – С. 50–51.

Депонированные рукописи

14–А. Петрик, С.Ю. Исследование эффективности обфускации исходных текстов программ (на языке C#) / С.Ю. Петрик; Белорус. гос. ун-т информатики и радиоэлектроники. – Минск, 2004 – 10 с. – Деп. в БелИСА 13.04.2004, № Д200431.

РЭЗЮМЭ

Петрык Сяргей Юр'евіч

Абарона праграмнага забеспячэння ад зваротнага праектавання з выкарыстаннем заблытваючага пераўтварэння

Ключавыя словы: праграмнае забеспячэнне, зваротнае праектаванне, заблытваючае пераўтварэнне. ***Аб'ект даследавання*** – праграмнае забеспячэнне. ***Прадмет даследавання*** – метады тэхнічнай абароны праграмнага забеспячэння ад несанкцыянаванага выкарыстання. ***Мэта працы*** – распрацоўка эфектыўных метадаў абароны праграмнага забеспячэння ад зваротнага праектавання і несанкцыянаванага даследавання.

Атрыманыя вынікі і іх навізна: распрацавана метрычная мадэль ацэнкі эфектыўнасці заблытваючага пераўтварэння на аснове метрык; ацэнена эфектыўнасць існуючых метадаў заблытваючага пераўтварэння: лексічнага метаду і метаду ўкаранення ў код галінаваных канструкцый; прапанаваны метад заблытваючага пераўтварэння, заснаваны на штучным павелічэнні счашлення модуляў праграмы, што дазваляе ўскладніць для зламыснікаў задачу выкарыстання цалкам асобных модуляў, без разумення іх логікі і выкарыстаных у іх алгарытмаў, праведзена ацэнка эфектыўнасці гэтага метаду; прадстаўлены алгарытм заблытвання даных, заснаваны на пераўтварэнні ўсіх пераменных модуляў у масівы базавага тыпу, што дазваляе схаваць сапраўдныя тыпы пераменных і павялічыць эфектыўнасць лексічнага метаду заблытваючага пераўтварэння, замінаючы аўтаматычнае прысваенне колькі-небудзь асэнсаваных імёнаў; распрацаваны праграмны комплекс абароны праграмнага забеспячэння сродкамі заблытваючага пераўтварэння Obfuscation Studio, які адрозніваецца павышанай здольнасцю да расшырэння і ўжо ўключае ў сябе модулі, якія рэалізуюць метады заблытваючага пераўтварэння, апісаныя ў дысертацыі.

Выкарыстанне і вобласць ужывання: распрацаваная сістэма абароны праграмнага забеспячэння метадамі заблытваючага пераўтварэння выкарыстоўваецца на прадпрыемствах «ЭПАМ Сістэмз» і «ПМіС-Софтпрадукт» для абароны распрацоўваемага праграмнага забеспячэння; атрыманыя ў рамках дысертацыйнай працы вынікі ўкаранёныя ў навучальны працэс, аб чым маюцца акты ўкаранення.

РЕЗЮМЕ

Петрик Сергей Юрьевич

Защита программного обеспечения от обратного проектирования с использованием запутывающего преобразования

Ключевые слова: программное обеспечение, обратное проектирование, запутывающее преобразование. ***Объект исследования*** – программное обеспечение. ***Предмет исследования*** – методы технической защиты программного обеспечения от несанкционированного использования. ***Цель работы*** – разработка эффективных методов защиты ПО от обратного проектирования и несанкционированного исследования.

Полученные результаты и их новизна: разработана модель оценки эффективности запутывающего преобразования на основе метрик; оценена эффективность существующих методов запутывающего преобразования: лексического метода и метода внедрения в код ветвящихся конструкций; предложен метод запутывающего преобразования, основанный на искусственном увеличении сцепления модулей программы, что позволяет усложнить для злоумышленников задачу использования целиком отдельных модулей, без понимания их логики и используемых в них алгоритмов, проведена оценка эффективности данного метода; представлен алгоритм запутывания данных, основанный на преобразовании всех переменных модулей в массивы базового типа, что позволяет скрыть истинные типы переменных и увеличить эффективность лексического метода запутывающего преобразования, затрудняя автоматическое присвоение сколько-нибудь осмысленных имен; разработан программный комплекс защиты ПО средствами запутывающего преобразования Obfuscation Studio, отличающийся повышенной способностью к расширяемости и уже включающий в себя модули, реализующие методы запутывающего преобразования, описанные в диссертации.

Использование и область применения: разработанная система защиты программного обеспечения методами запутывающего преобразования используется на предприятиях «ЭПАМ Системз» и «ПМиС-Софтпродукт» для защиты разрабатываемого программного обеспечения; полученные в рамках диссертационной работы результаты внедрены в учебный процесс, о чём свидетельствуют имеющиеся акты внедрения.

SUMMARY

Petryk Siarhei Yur'evich

Software protection against reverse engineering by the means of obfuscation

Keywords: software, reverse engineering, obfuscation. ***Object of research*** is software. ***Subject of research*** is methods of software protection form unauthorized use. ***The purpose of work*** is development of effective methods of software protection against reverse engineering and unauthorized use.

Received results and their novelty: metric model for obfuscation efficiency estimation is developed; efficiency of such obfuscation transformations as lexical and predicates inserting is estimated; new obfuscation method based on artificial modules coupling increasing is proposed, this method makes it harder to use modules of a program without investigating their logic, the efficiency of this method is also estimated; new method of data manipulation is developed, it is based on casting all the variables to a basic type that hides real type of variables and therefore makes it extremely hard to reverse lexical transformations; obfuscator «Obfuscation Studio» is developed that is characterized by high potency to extendibility and includes obfuscating modules that are described in dissertation.

Utilization and field of application: developed software defense system is used to protect the software, which is produced by «EPAM Systems» and «PM&S-Softproduct» enterprises; the results of research are being applied in educational process that is documented by corresponding utilization acts.

ПЕТРИК Сергей Юрьевич

ЗАЩИТА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ОТ ОБРАТНОГО ПРОЕКТИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ЗАПУТЫВАЮЩЕГО ПРЕОБРАЗОВАНИЯ

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук
по специальности 05.13.19 – Методы и системы защиты информации, информационная безопасность

Подписано в печать 15.05.2008. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 1,63.
Уч.-изд. л. 1,2. Тираж 60 экз. Заказ 285.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6