УДК 681.3.06

А. В. Мазин, Д. А. Щелкунов

ИССЛЕДОВАНИЕ И РАЗРАБОТКА НОВОГО МЕТОДА ОБФУСКАЦИИ

Рассмотрен метод построения алгоритмов обфускации, обеспечивающий повышенную стойкость программных продуктов и данных от злоумышленников. Даны определения стойкости подпрограмм по отношению к автоматическим средствам деобфускации, определены их функциональные свойства. Сформулирована и доказана теорема о NP-полноте задачи деобфускации.

Ключевые слова: *кодирование информации, программа, криптография, запутывание кода, алгоритм, оптимизация.*

Теоретические аспекты запутывания кода по Бараку заключаются в том, что вероятностный алгоритм O, рассматриваемый как машина Тьюринга (ТМ), является обфускатором при выполнении следующих трех условий [1]:

1. *Функциональность*. Для любой машины Тьюринга M алгоритм $O(M)$ вычисляет ту же функцию, что и M.

2. *Полиномиальное замедление*. Длина и время выполнения алгоритма $O(M)$ должны быть полиноминально больше аналогичных показателей для M, т.е. существует такой полином p, что $|O(M)| < p(|M|)$, где $|M|$ — размер программы M, а $|O(M)|$ — размер запутанной программы $O(M)$, и если M заканчивает свою работу через t шагов при определенном входе x, то $O(M)$ оканчивает свою работу за $p(t)$ шагов при том же входе x.

3. *Свойство виртуальной Black-Box*. Для любой вероятностной машины Тьюринга PPT A найдется вероятностная машина Тьюринга PPT S, такая, что для всех ТМ M справедливо

$$\left|Pr\left[A(O(M)) = 1\right] - Pr\left[S^{\langle M\rangle}(1^{|M|}) = 1\right]\right| \leqslant neg(|M|),$$

где Pr — вероятность соответствующего события, neg — бесконечно малая величина.

Новый метод обфускации должен обладать следующими свойствами:

- возможностью работы с готовыми к запуску приложениями (машинное представление),
- более высоким, чем у виртуальных машин, быстродействием при схожем уровне защищенности,
- возможностью импортирования в другие программные и аппаратные платформы,
- возможностью привязки графа потока управления запутанной подпрограммы к некоторому внешнему ресурсу.

Разработанный метод обфускации подразумевает добавление фальшивого алгоритма к истинному алгоритму в целях максимального затруднения выявления истинного алгоритма в чистом виде. На практике наиболее ценным является свойство стойкости к автоматическим алгоритмам оптимизации или деобфускации. Этим свойством имеет смысл заменить более строгое свойство Black-Box по Бараку. Формально это свойство выглядит так [2]:

$$Pr\left[OptA\lg(O(M)) = 1\right] \leqslant neg(|M|), \tag{1}$$

где *OptAlg* — оптимизирующий алгоритм.

Такое свойство означает, что запутанная подпрограмма $O(M)$ считается стойкой к алгоритму оптимизации, если в результате применения преобразований *OptAlg* по отношению к подпрограмме $O(M)$ исходная подпрограмма M не будет восстановлена [3]. Оптимизирующий алгоритм *OptAlg* должен являться детерминированной ТМ и находиться в классе сложности P. Если *OptAlg* является недетерминированной ТМ, то распознавание языка $O(M)$ за полиномиальное время — вполне разрешимая задача. По сути дела, это накладывает обязательность интерактивности процесса оптимизации запутанного кода и данных, а также подразумевает то, что алгоритм *OptAlg* должен лежать в классе NP. Неформально говоря, условие (1) означает, что достаточно обеспечить невозможность создания автоматического алгоритма деобфускации (оптимизации) для класса запутывающих преобразований O.

С учетом изложенного приведем следующие определения.

Определение 1. Запутанная подпрограмма является оптимизационно стойкой, если не существует алгоритма оптимизации этой подпрограммы, лежащего в классе сложности P.

Определение 2. Запутанная подпрограмма $O(M)$ является оптимизационно стойкой по отношению к алгоритму оптимизации *OptAlg*, если справедлива формула (1).

В принципе, определение, заданное формулой (1), можно сделать менее строгим, всего лишь ограничив универсальность алгоритма *OptAlg*:

$$\begin{aligned} &if(Pr\left[OptAlg(O(M)) = 1\right] > neg(|M|)), \\ &(Pr\left[OptAlg(O'(M))=1\right] \approx Pr\left[OptAlg(O'(M'))=1\right]) \leqslant neg(|M|). \end{aligned} \tag{2}$$

В данном случае $O'(M)$ — это еще одно применение алгоритма O по отношению к подпрограмме M, а $O'(M')$ — это применение алгоритма O по отношению к подпрограмме M', отличающейся от M. Таким образом, даже если возможно построить алгоритм оптимизации *OptAlg*, то он будет актуален только для конкретной подпрограммы $O(M)$.

Пример программы, содержащей запутанный код

Определение, заданное формулой (2), является менее строгим, чем (1) и ограничивает лишь универсальность преобразований *OptAlg*, что наиболее соответствует практической задаче обфускации коммерческих приложений.

Рассмотрим замкнутую систему, изображенную на рисунке: $O(M)$ — подпрограмма, к которой были применены запутывающие преобразования; A — подпрограмма, вызывающая подпрограмму $O(M)$; входные данные — вектор X размерностью p; выходные данные — вектор Y размерностью q.

Дадим определение функционального свойства подпрограммы.

Определение 3.

$$\forall p_1, p_2 \in \Pi(f_{p_1} = f_{p_2} \Rightarrow (p_1 \in \pi \Leftrightarrow p_2 \in \pi)), \tag{3}$$

где $\pi \subseteq \Pi$ — функциональное свойство подпрограммы; Π — множество всех подпрограмм, удовлетворяющих условию эффективности (выполняющихся за полиномиальное время); p_1, p_2 — подпрограммы, принадлежащие множеству Π; f_{p_1}, f_{p_2} — функции, вычисляемые соответственно подпрограммами p_1, p_2.

Введем левостороннюю операцию “*”, называемую умножением функциональных свойств и фактически являющуюся последовательным выполнением инструкций с этими функциональными свойствами.

Эта операция будет иметь следующие свойства.

Свойство 1.

$$\pi_1 * \pi_2 \neq \pi_2 * \pi_1.$$

Свойство 2. Существование единичного элемента.

Единичным элементом e на множестве функциональных свойств Π будет являться функциональное свойство такое, что

$$\pi * e = e * \pi = \pi.$$

Свойство 3. Ассоциативность.

$$(\pi_1 * \pi_2) * \pi_3 = \pi_1 * (\pi_2 * \pi_3).$$

Следует особо оговорить смысл функциональных свойств. Далее под функцией будем понимать некоторый алгоритм, который получает на вход некоторое число векторов, называемых векторами входных

значений, а результат внутренних преобразований записывает в некоторое число векторов, называемых векторами выходных значений. Рассмотрим функцию $F(X, Y)$, где X и Y — векторы входных и выходных значений. Эту функцию можно представить в следующем виде:

$$F(X,Y) = f_1(X, V_0, Y_0, V_1, Y_1) * f_2(X, V_1, Y_1, V_2, Y_2) * \ldots$$

$$\ldots * f_n(X, V_{n-1}, Y_{n-1}, V_n, Y_n), \quad (4)$$

где X_i и Y_i — векторы входных и выходных значений; V_i — вектор промежуточных значений.

Коэффициенты при векторах необходимы для подчеркивания итеративности и отделения входных параметров от выходных. Входные векторы V_{i-1}, Y_{i-1} функции f_i являются выходными векторами функции f_{i-1}, так же как выходные векторы V_i и Y_i функции f_i являются входными для функции f_{i+1}. По сути дела, существуют только один вектор V и один вектор Y, значения элементов которых меняются от функции к функции. Операция "*", примененная в формуле (4), аналогична операции умножения функциональных свойств согласно определению функционального свойства. Рассмотрим эти свойства по отношению к функциям f_i.

Свойство 1. Операция "*" некоммутативна.

Свойство 2. Функция $f_i(X, V_{i-1}, Y_{i-1}, V_i, Y_i)$ называется единичной (e), если справедлива система

$$\begin{cases} V_i = V_{i-1}; \\ Y_i = Y_{i-1}. \end{cases}$$

Свойство 3. Операция "*" ассоциативна, т.е.

$$f_i(X, V_{i-1}, Y_{i-1}, V_i, Y_i) * \\ * [f_{i+1}(X, V_i, Y_i, V_{i+1}, Y_{i+1}) * f_{i+2}(X, V_{i+1}, Y_{i+1}, V_{i+2}, Y_{i+2})] = \\ = [f_i(X, V_{i-1}, Y_{i-1}, V_i, Y_i) * f_{i+1}(X, V_i, Y_i, V_{i+1}, Y_{i+1})] * \\ * f_{i+2}(X, V_{i+1}, Y_{i+1}, V_{i+2}, Y_{i+2}).$$

Если бы для каждой f_i существовал единственный обратный элемент f_i', такой, что $f_i * f_i' = f_i' * f_i = e$, то семейство функций f_i можно было бы назвать группой по операции "*", однако не для каждой f_i существует обратный элемент.

Остановимся более подробно на понятии обратного элемента и обратимости в целом.

Определение 4. Функция f называется обратимой, если существует такая функция f', что $f * f' = f' * f = e$.

Функция f' должна "уметь" восстановить по выходным векторам функции f и вектору X входные векторы функции f. Рассмотрим

теперь функцию $f_i(X, V_{i-1}, Y_{i-1}, V_i, Y_i)$. Предположим, что единственной ее задачей является копирование входных данных в выходные: $Y = V = X$. Предположим также для упрощения, что размерность векторов X, Y, V одинакова. Совершенно очевидно, что для данной функции невозможно найти такую функцию $f_{i+1}(X, V_i, Y_i, V_{i+1}, Y_{i+1})$, которая за полиномиальное время находила бы векторы V_{i+1}, Y_{i+1}, такие, что $V_{i+1} = V_{i-1}$ и $Y_{i+1} = Y_{i-1}$, т.е. для функции, приведенной ранее, не существует обратной.

Очевидно, чтобы функция $f_i(X, V_{i-1}, Y_{i-1}, V_i, Y_i)$ была обратима, необходимо, чтобы значение каждого из элементов выходных векторов V_i, Y_i зависело от соответствующих элементов векторов V_{i-1}, Y_{i-1}. Однако это требование не является достаточным. Зависимость должна быть такой, чтобы обратные вычисления могли быть проведены за полиномиальное время.

Все сказанное о функциях справедливо и для функциональных свойств по определению.

Теперь изучим детально процесс оптимизации запутанного кода. Рассмотрим равенство

$$\pi_{\text{исх}} = \pi_1 * \pi_2 * \ldots * \pi_n, \tag{5}$$

где $\pi_{\text{исх}}$ — функциональное свойство подпрограммы M до применения запутывающих преобразований; π_i — элементарные свойства подпрограммы, из которых складывается $\pi_{\text{исх}}$; $i = [1 \ldots]$.

Приведем определение элементарных свойств подпрограммы.

Определение 5. Элементарное свойство подпрограммы — это свойство подпрограммы, соответствующее одной трехадресной инструкции.

Определение 6. Произведение элементарных свойств подпрограммы называется оптимальным по отношению к указанному алгоритму оптимизации, если применение к этому произведению указанного алгоритма не изменяет сомножителей произведения.

Предположим, что правая часть равенства (5) содержит оптимальное произведение свойств подпрограмм по отношению к определенному алгоритму оптимизации (алгоритму A).

После применения запутывающих преобразований, делающих последовательность из правой части равенства (5) неоптимальной по отношению к алгоритму A, получаем:

$$\pi_{\text{исх}} = \pi_1 * \nu_1 * \pi_2 * \nu_2 * \ldots * \pi_n * \nu_n \tag{6}$$

или

$$\pi_{\text{исх}} = \pi_0 * \nu_0 * \pi_1 * \nu_1 * \pi_2 * \nu_2 * \ldots * \pi_n * \nu_n, \tag{7}$$

где $\nu_0, \nu_1, \nu_2, \ldots, \nu_n$ — добавленные функциональные свойства; $\pi_0 = e$.

Отметим, что если $\nu_0, \nu_1, \nu_2, \ldots \nu_n$ из формулы (7) равняются e, то равенство (7) будет сводиться к системе равенств следующего вида:

$$\begin{cases} \pi_0 = \nu_0 = e; \\ \pi_1 = \pi_1 * \nu_1; \\ \pi_2 = \pi_2 * \nu_2; \\ \cdots\cdots\cdots\cdots \\ \pi_n = \pi_n * \nu_n. \end{cases} \tag{8}$$

Действительно, компоненты формулы (7) в данном случае могут анализироваться алгоритмом оптимизации независимо друг от друга. Таким образом, существенно упрощается процедура анализа, а следовательно, повышается вероятность существования алгоритма оптимизации, лежащего в классе сложности P.

На самом деле, несводимость равенства (7) к системе (8) можно обеспечить. Для этого, прежде всего, необходимо, чтобы подпрограммы с функциональными свойствами π_i и ν_i определяли разные элементы выходных векторов, а подпрограмма с функциональным свойством ν_{k-1} восстанавливала необходимые элементы входных векторов перед тем, как будет запущена подпрограмма с функциональным свойством π_k, которая будет работать непосредственно с этими элементами ($k > n$). Однако такой подход не обладает высокой стойкостью по интуитивно понятным причинам. Есть еще одна возможность обеспечить несводимость равенства (7) к системе (8). Далее приведено применение алгоритмов гомоморфного шифрования или вычисления над зашифрованными данными:

$$m_1 \, op \, m_2 \Leftrightarrow E(m_1) \, op' \, E(m_2), \tag{9}$$

т.е. определенной операции над двумя исходными текстами $m_1 \, op \, m_2$ взаимно однозначно соответствует другая операция над зашифрованными данными $E(m_1) \, op' \, E(m_2)$. Однако известно, что формула (9) выполнима не для всех операций op.

Рассмотрим еще раз последовательность в правой части равенства (7). Предположим, что все ее элементы обратимы. Тогда справедливо следующее:

$$\nu_0 = \pi * \nu'_n * \pi'_n * ... * \nu'_1 * \pi'_1, \tag{10}$$

т.е.

$$\pi = \pi * \gamma, \tag{11}$$

где

$$\gamma = \nu'_n * \pi'_n * \ldots * \nu'_1 * \pi'_1 * \pi_1 * \nu_1 * \ldots * \pi_n * \nu_n. \tag{12}$$

Из равенства (9) следует, что $\gamma = e$, т.е. злоумышленнику достаточно будет выделить свойства π и γ.

Однако если элементы из правой части равенства (7) необратимы, то достаточно сложно создать алгоритм, который автоматически генерировал бы $\nu_0 \dots \nu_n$ таким образом, чтобы они, удовлетворяя равенству (7), не сводились к системе (8).

Итак, из всего сказанного ранее можно сделать вывод, что при автоматической генерации подпрограмм с функциональными свойствами $\nu_0 \dots \nu_n$ мы не можем гарантированно утверждать, что не существует эффективного алгоритма оптимизации, позволяющего восстановить исходную последовательность (5).

Предположим теперь, что

$$\pi = \pi_0 * \nu_0 * \pi_1 * \nu_1 * \pi_2 * \nu_2 * \ldots * \pi_n * \nu_n \neq \pi_{\text{исх}}, \qquad (13)$$

т.е. подпрограмма $O(M)$ обладает функциональным свойством, отличным от функционального свойства подпрограммы M. Этого можно добиться посредством введения глобального по отношению к подпрограмме $O(M)$ контекста. Если подпрограммы с функциональными свойствами $\nu_0 \dots \nu_n$ будут работать с фальшивым контекстом, сохраняя в нем результат, то достаточно сложно будет отделить истинные данные от ложных, не анализируя при этом подпрограмму A. Очевидно, что несводимость равенства (7) к системе (8) тоже достаточно легко обеспечить. Для достижения желаемого злоумышленником эффекта алгоритмы оптимизации нужно будет теперь применять по отношению к подпрограммам A и $O(M)$.

Теорема. Задача определения существенности функционального свойства $\pi_i(\nu_i)$ в равенстве (13) NP-полна.

Доказательство. Докажем сначала сводимость задачи определения существенности функционального свойства к задаче выполнимости булевой формулы. Для того чтобы проверить существенность некоторого функционального свойства (т.е. влияние его наличия на результат работы программы), необходимо исключить это функциональное свойство из последовательности (13) и проверить результаты выполнения подпрограммы на всех входных наборах, т.е., по сути, выполнить булеву формулу следующего вида:

$$\bigcup_i (X_i \cdot \overline{Y_i}), \qquad (14)$$

где X и $\overline{Y}$ — выходные данные, полученные до и после исключения функционального свойства.

Если результат формулы (14) не будет нулевым, то исключенное функциональное свойство будет существенным. Очевидно, что задача определения существенности функционального свойства сводится к задаче выполнимости булевской формулы, а следовательно, лежит в классе NP.

Чтобы проверить выполнимость булевой формулы, необходимо проверить ее значение некоторое число раз, т.е. необходимо проверить значения существенных составляющих булевой формулы. Таким образом, можно утверждать, что задача выполнимости булевой формулы сводится к задаче определения существенности ее составляющих (т.е., по сути, проверить существенность функциональных свойств в нашем контексте определений) [4].

С учетом изложенного выявили, что задача определения существенности функциональных свойств в равенстве (13) NP-полна.

Итак, подведем итог приведенным теоретическим соображениям. Во-первых, как доказал Барак в своих работах, обфускация в общем случае невозможна, так как существует класс программ, для которых условие Black-Box невыполнимо. Во-вторых, даже если запутываемая подпрограмма не включена в класс программ, неподверженных обфускации по Бараку, то сгенерированный автоматически алгоритм запутывания, при условии выполнения равенства (7) будет либо сводиться к равенству (11), что по вполне понятным причинам нежелательно, либо к системе (8). Следует особо подчеркнуть, что сводимость к системе (8) вовсе не означает, что "взлом" будет тривиальным, ведь функциональные свойства $\pi_0, \pi_1 \dots \pi_n$ могут быть реализованы подпрограммами с достаточно высокими метриками сложности.

Однако, как уже было указано ранее, единожды проведенный статический или полустатический анализ запутанного кода позволит выявить основное функциональное свойство подпрограммы и определить его составляющие. Впоследствии возможно создание автоматических или полуавтоматических средств, позволяющих производить полную или частичную оптимизацию (деобфускацию). В любом случае, эффект Black-Box при этом не проявляется.

Добавление глобального (по отношению к исследуемой подпрограмме) контекста обеспечит гораздо более высокую стойкость запутанного кода по отношению к существующим алгоритмам оптимизации. Усложнится при этом и статический (полустатический) анализ подпрограммы. Если следовать при этом ряду рекомендаций по построению запутывающих преобразований, то можно существенно снизить вероятность построения деобфускатора, работающего за полиномиальное время.

Следует заметить особо, что приведенная доказанная ранее теорема справедлива только в том случае, если нет существенных различий в подпрограммах, реализующих истинные и фальшивые функциональные свойства. Например, если инструкции исходной подпрограммы в качестве операндов имели числа с плавающей точкой, а добавленные инструкции работают только с целыми числами, то в данной ситуации

вполне возможно будет отделить истинные инструкции от фальшивых автоматически за полиномиальное время.

Снова встает вопрос о соответствии данного подхода свойству Black-Box по Бараку. Очевидно, что если подпрограмма A недоступна, то доказательство Барака в данном случае не работает.

Если подпрограмма A (см. рисунок) доступна, то метод теряет лишь часть своей стойкости. Злоумышленнику необходимо будет доказать, что только подпрограмма A вызывает подпрограмму $O(M)$, что не всегда представляется возможным. В частных случаях деобфускация все-таки может быть осуществлена, но она потребует несравненно бóльших усилий и анализа всего кода приложения в отличие от случая, когда принцип функциональности по Бараку соблюдается. В лучшем случае удастся получить соответствие формулам (1) или (2), что вполне допустимо в практических целях.

СПИСОК ЛИТЕРАТУРЫ

1. B a r a k B. Non-Black-Box Techniques in Cryptography., Thesis for the Ph.D. Degree, Department of Computer Science and Applied Mathematics. The Weizmann Institute of Science. January 6, 2004.
2. Щ е л к у н о в Д. А. Обфускация. Теоретические и практические аспекты // Тр. междунар. конф. РусКрипто. Февраль, 2007.
3. Щ е л к у н о в Д. А. Запутывание программ и внедрение в приложение стороннего кода. Технологии Microsoft в теории и практике программирования // Тр. Всеросс. конф. студентов, аспирантов и молодых ученых, Москва. 2–3 апреля, 2007, МАИ.
4. М а з и н А. В., Щ е л к у н о в Д. А. Анализ и применение запутывающих преобразований при защите исполняемых файлов от несанкционированного использования // Прогрессивные технологии, конструкции и системы в приборо- и машиностроении. Материалы. Т. 1. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005.

Статья поступила в редакцию 24.03.2008

Анатолий Викторович Мазин родился в 1948 г., окончил Одесское высшее инженерное морское училище в 1972 г. Канд. техн. наук, доцент кафедры “Компьютерные системы и сети” Калужского филиала МГТУ им. Н.Э. Баумана. Автор 46 научных работ в области информационных технологий и технической диагностики.

A.V. Mazin (b. 1948) graduated from the Odessa Higher Engineering Nautical School in 1972. Ph. D. (Eng.), assoc. professor of “Computer Systems and Networks” department of the Kaluga Branch of the Bauman Moscow State Technical University. Author of 46 publications in the field of information technologies and technical diagnostics.

Дмитрий Анатольевич Щелкунов родился в 1980 г., окончил Калужский филиал МГТУ им. Н.Э. Баумана. Аспирант кафедры “Компьютерные системы и сети” Калужского филиала МГТУ им. Н.Э. Баумана. Автор 10 научных работ в области информационных технологий и информационной безопасности.

D.A. Shchelkunov (b. 1980)) graduated from the Kaluga Branch of the Bauman Moscow State Technical University in 2007. Post-graduate of “Computer Systems and Networks” department of the Kaluga Branch of the Bauman Moscow State Technical University. Author of 10 publications in the field of information technologies and technical diagnostics.