Д.А. Щелкунов
Компания «Актив»

Применение запутывающих преобразований и полиморфных технологий для автоматической защиты исполняемых файлов от исследования и модификации

В настоящее время существует целый ряд подходов к защите исполняемых файлов от исследования и модификации. Основные из них – это применение конвертов и виртуализация кода приложения. При этом код конверта или код тела виртуальной машины (ВМ) осуществляет необходимые проверки валидности данной копии приложения. Можно легко увидеть, что качество практически всех подходов, применяемых для защиты ПО, напрямую зависит от качества запутывающих преобразований. По сути дела виртуализация кода – это уже запутывающие преобразования. В этом случае необходимо применить запутывающие преобразования по отношению к телу ВМ для того, чтобы во-первых – воспрепятствовать её изучение и во-вторых – минимизировать возможность создания декомпиляторов для данной ВМ. Существует два направления развития технологии обфускации. Первое – чисто теоретическое. Определение идеального обфускатора было дано Б. Бараком в своих тезисах [1].

Вероятностный алгоритм *O* является *TM* обфускатором при выполнении следующих трех условий:
1) Функциональность. Для любой *TM M O(M)* вычисляет ту же функцию, что и *M*.
2) Полиномиальное замедление. Длина и время выполнения *O(M)* должна быть полиноминально больше аналогичных показателей для *M*. Т.е существует полином *p* такой, что и если *M* заканчивает свою работу через *t* шагов при определенном входе *x*, то *O(M)* оканчивает свою работу за не более чем *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 *TM M* справедливо:

$$\left|Pr\ A(O(M))=1 - Pr\ S^{\langle M\rangle}(1^{|M|})=1\right| \le neg(|M|) \qquad (1)$$

Т.е., говоря неформально, не существует алгоритма распознавания обфускации более эффективного, чем обычное предположение сделанное на основе анализа входов и выходов запутанной программы (функции или запутанного набора инструкций).

Бараком было доказано, что универсального обфускатора не существует, т.к. существует класс подпрограмм, для которых основное функциональное свойство устанавливается достаточно легко. В работах [2], [3], [4] и [5] рассказывалось о технологии обфускации на уровне 3-х адресного кода и о технологии перевода из машинного кода в 3-х адресный. В работах [4] и [5] было показано, что при нарушении принципа функциональности по Бараку можно добиться достаточно стойкой обфускации. Возможно даже будет соблюдаться свойство виртуальной Black-Box. Т.к. разработанные алгоритмы обфускации подразумевают их применение по отношению к готовым исполняемым файлам, то вместо свойства Black-Box было введено свойство стойкости по отношению к алгоритмам оптимизации.

$$\begin{aligned} &if(Pr\ OptAlg(O(M))=1 \ge neg(|M|)) \\ &\quad Pr\ OptAlg(O'(M))=1 \approx Pr\ OptAlg(O'(M'))=1 \le neg(|M|) \end{aligned} \qquad (2)$$

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*. Т.е. даже если возможно построить алгоритм оптимизации *OptAlg*, то он будет актуален только для конкретной подпрограммы *O(M)*. Для того, чтобы удовлетворять этому свойству, необходимо следовать правилам, приведенным в работах [4] и [5].

Разработанная технология обфускации подразумевает 2-х уровневую обфускацию, т.е. вначале производится запутывание на уровне 3-х адресного кода, а затем осуществляется обфускация на уровне конкретной целевой платформы. Для обфускации на уровне целевой платформы используются алгоритмы генерации полиморфного кода, удовлетворяющие следующим условиям:

$$\begin{aligned} &instr_i \to p_i : p_i, instr_i \in P_i^{<k>},\ P_i^{<k>} \subset \pi_i \\ &instr_{i+1} \to p_{i+1} : p_{i+1}, instr_{i+1} \in P_{i+1}^{<k>},\ P_{i+1}^{<k>} \subset \pi_{i+1} \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned} \qquad (3)$$

$$instr_{i+m} \rightarrow p_{i+m} : p_{i+m}, instr_{i+m} \in P_{i+m}^{<k>},\ P_{i+m}^{<k>} \subset \pi_{i+m}$$

$$\{instr_i, instr_{i+1}, ..., instr_{i+m}\} \rightarrow p : p \subset \pi_i \cdot \pi_{i+1} \cdot ... \cdot \pi_{i+m}$$

$$p \neq W(p_i, p_{i+1}, ..., p_{i+m})$$

Вышеприведенная система условий означает, что каждой инструкции поставлен в соответствие некий набор подпрограмм, обладающих одинаковым функциональным свойством (делающих одно и то же по разному). Назовем их элементарными. Однако полученная в результате программа p не должна быть последовательностью элементарных подпрограмм. Такую технологию можно назвать технологией пересечения полиморфных инструкций. Например, часть инструкций подпрограммы p_2 переносится в подпрограмму p_1 или наоборот. Это можно сделать при условии, что подпрограммы p_1 и p_2 не работают с общими данными, а точнее сказать, беспрепятственно можно переместить только те инструкции, которые работают с операндами отличными от операндов инструкций подпрограммы, в которую осуществляется перенос. Особое внимание следует уделить арифметическим инструкциям XOR, OR, AND, ADD, SUB, NOT. Эти инструкции можно генерировать полиморфно с использованием различных базисов, например, штрих Шеффера и стрелка Пирса. Активно применяются также законы булевой алгебры такие как закон де Моргана и закон поглощения. При генерации полиморфных эквивалентов инструкций есть одна проблема – учет флагов. Дело в том, что если флаги, изменяемые данной инструкцией учитываются где-то в программе, то сгенерировать её полиморфный эквивалент гораздо сложнее и, напротив, гораздо легче понять, что это за инструкция. Поэтому в разработанном генераторе полиморфного кода есть 2 основных режима генерации – с учетом флагов и без него. Если есть подозрение, что в программе используются флаги, определяемые данной инструкцией, то она генерируется в режиме с учетом флагов. В противном случае инструкция генерируется без учета флагов. Генератор полиморфного кода используется, в частности, для защиты тела BM Guardant.

Литература:

1) Barak 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) Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии, инструменты. : Пер. с англ. – М. : Издательский дом «Вильямс», 2003. – 768 с. ил.;
3) Gogul Balakrishnan, Thomas Reps. Analyzing Memory Accesses in x86 Binary Executables.
4) Щелкунов Д.А. Методы автоматической защиты Windows-приложений от исследования и несанкционированной модификации., Технологии Microsoft в теории и практике программирования. Труды Всероссийской конференции студентов, аспирантов и молодых ученых., Москва 2-3 марта 2006 г., МГТУ им. Н.Э. Баумана;
5) Щелкунов Д.А. Обфускация. Теоретические и практические аспекты; доклад на конференции РусКрипто-2007.