УДК 004.058.5

ЗАПУТЫВАЮЩЕЕ КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Исаев Д.
Костанайский государственный университет им.А.Байтурсынова, Костанай

Научный руководитель: Махамбетова Г. И. ст. преподаватель кафедры программного обеспечения

Программное обеспечение (ПО) современных компьютерных систем представляет собой сложное изделие, и обеспечение абсолютного качества программных продуктов (ПП) представляет собой практически неразрешимую задачу, поэтому необходимость внесения защитных функций на всем протяжении ЖЦ ПО не вызывает сомнений.

Существует три вида атак на ПП: воровство (piracy), обратная инженерия (reverse engineering) и модификация кода (code modification). В большинстве случаев, для осуществления атаки взломщику требуется изучить принцип работы кода программы – этот процесс изучения называется процессом обратной инженерии (понимания).

Обфускация ("obfuscation" – запутывание) – это одно из достаточно универсальных программных средств (ПС) защиты программного кода, которое позволяет усложнить процесс обратной инженерии кода защищаемого ПП и тем самым значительно повысить временные затраты злоумышленника на проведение атаки [1]. Для сравнительно быстрого приведения исходного кода программы в нечитабельное состояние, применяются различные методы лексической обфускации – вида обфускации программного кода, заключающегося в форматировании кода программы, изменении его структуры таким образом, чтобы он стал нечитабельным, трудным для изучения и менее информативным, и обфускации управления, заключающейся в запутывании управляющей логики программы [1].

С каждым днем на рынке ПО появляются более совершенные обфускаторы, которые оперируют различными алгоритмами и, в зависимости от этого, предоставляют разную степень обфусцированности программы, но и обладают специфическими недостатками, такими, например, как как отсутствие теоретических исследований, на основе которых реализуются различные методы запутывания, отсутствие оценки эффективности работы преобразователей и потерь во времени выполнения и в объеме программ при их запутывании, а также недостаточная гарантированность функциональной эквивалентности исходной и запутанной программы.

С учетом недостатков существующих обфускаторов было разработано новое ПС для обфускации программного кода, в котором путем анализа быстродействия работы и оценки эффективности преобразования были совмещены различные методы лексической обфускации и обфускации управления: замена пользовательских имен идентификаторов на неинформативные имена, объединение и увеличение времени жизни некоторых переменных, удаление комментариев и отметок типа #region <имя_функции> и #endregion, использованных в коде для лучшего понимания его функциональности, форматирование кода – удаление пробелов и отступов в коде, использованных для лучшего восприятия кода, а также ввод фиктивных предикатов для усложнения понимания управляющей логики программы. Разработанное ПС выгодно отличается от ему подобных стоимостью преобразования и выполнения кода после обфускации – запутанная программа незначительно увеличивается в объеме и выполняется практически с той же скоростью. Также в начале работы определяется оценка целесообразности применения обфускации для исходного кода программы, путем вычисления объема потенциально запутываемого кода.

Теория обфускации на сегодняшний день находится на стадии формирования. До сих пор не формализованы метрики, позволяющие напрямую определить степень эффективности обфускатора, и следовательно запутанности кода программы. Поэтому оценить данную эффективность можно косвенно, используя классические метрики сложности ПО, оценивающие сложность программы в целом, например: число строк кода, процент комментариев в общем объеме кода, цикломатическая сложность.

Существует возможность вычислить цикломатическую сложность напрямую из исходного кода. Это обеспечивает возможность измерения и управления сложностью во время разработки, так как сложность может быть измерена даже до того, как модуль будет логически закончен. Большинство языковых конструкций добавляют фиксированное значение к сложности. Начиная со сложности, равной единице, конструкция выбора с M выходами добавляет (M – 1) к сложности, так как количество дуг «e» в формуле e - n + 2 расчета сложности увеличивается на (M – 1), в то время как количество вершин «n» не изменяется. Так, “if”, “while” и другие бинарные конструкции выбора добавляют единицу к сложности. А вклад в сложность конструкций множественного выбора типа “switch” – это точное число утверждений с меткой “case”, учитывая, что значение по умолчанию может быть не указано явно [2].

Логические операторы добавляют или единицу, или ничего к сложности. Так, например, оператор “&&” языка C# добавляет единицу к сложности, а оператор “and” языка Ада ничего не добавляет к сложности, так как он определен таким образом, что не генерирует вершину условия выбора на потоковом графе [2].

Таким образом, цикломатическая сложность для языка C# может быть определена как значение сложности, полученное в зависимости от числа конструкций выбора и операторов в программе.

Для оценки сложности кода программы вводится функция, которая оценивает значения рассчитанных метрик до и после обфускации, нормализует их и вычисляет расстояние Соренса между этими значениями, которое и является конечным значением сложности. Учитывая, что одни метрики подлежат максимизации, а другие минимизации, можно разделить их по этому критерию на 2 группы, и сделать вывод, что цель обфускации – получение максимального квадратического расстояния между этими группами метрик.

Литература

1. Collberg C., Thomborson C., Low D. A Taxonomy of Obfuscating Transformations. – Department of Computer Science, University of Auckland, 1997.

2. T. J. McCabe. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. Computer Systems Laboratory, NIST Gaithersburg, MD 20899-0001, 1996.