М. О. Шудрак, И. А. Лубкин, В. В. Золотарев

МЕТОДИКА ДЕКОМПИЛЯЦИИ БИНАРНОГО КОДА И ЕЕ ПРИМЕНЕНИЕ В СФЕРЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Введение

На сегодняшний день применение декомпиляция бинарного кода в сфере информационной безопасности ограничено ввиду наличия в этом процессе ряда фундаментальных проблем.

Анализ бинарного кода является нетривиальной задачей в силу того, что в процессе компиляции исходного кода программы теряется важная с точки зрения анализа информация [1]. Таким образом, полученная в ходе декомпиляции информация часто не представляет практического интереса.

Однако декомпиляция занимает важное место в сфере информационной безопасности и может быть успешно использована в следующих областях:

- защита программного обеспечения;
- анализ вредоносных объектов;
- поиск уязвимостей.

В данной статье авторы предлагают новый подход к декомпиляции бинарного кода и его применение к областям, которые описаны выше.

Декомпиляция кода

Процесс анализа бинарного кода требует некоторого уровня его абстракции от машинного кода к человеку [2]. Таким образом, основная идея описываемой методики заключается в том, что декомпиляция бинарного кода не требует представления машинного кода на языке высокого уровня, достаточно восстановить алгоритм функционирования участка анализируемого кода (тем самым добиться соответствующего уровня абстракции).

Методика интерпретации инструкций, предложенная авторами данной статьи, рассматривает инструкции как совокупность элементарных операций над ресурсами, где в качестве ресурса могут выступать регистры, память или регистры флагов процессора. Алгоритм работы декомпилятора представлен на рис. 1.

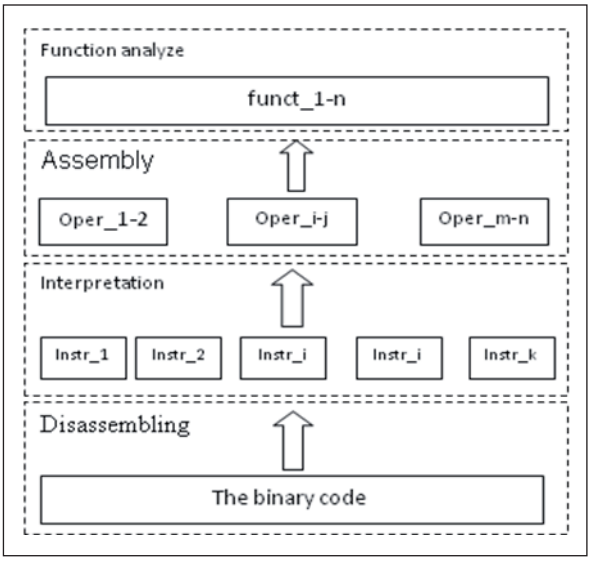

Рис. 1. Алгоритм анализа кода

Все операции в рамках методики были сгруппированы в общий базис, описывающий все множество возможных инструкций (таблица 1).

Таблица 1. Описание операций

Operation	**Description**
+ , –	Сложение и вычитание
*	Умножение
. /.	Целочисленное деление
%	Взятие остатка от деления
And	Логическое «и»
or	Логическое «или»
Xor	Логический «хог»
<<	Циклический сдвиг влево
>>	Циклический сдвиг вправо
[]	Доступ к отдельным битам
[<–	Запись в память по определенному адресу
<]	Чтение из памяти по определенному адресу
NOT	Логическое «НЕ»
«Transit»	Пересылка информации от одного ресурса к другому
«Const»	Обозначение констант
«Special».	Обозначение специальных инструкций

Для отображения информации об инструкции предлагается следующая схема: ресурс, соответствующий регистру, обозначается фигурными скобками, в которые заключен его номер. Ресурс оперативной памяти обозначается квадратными скобками, в которые заключен адрес ресурса. Номер такого ресурса выносится за скобки. Результат отделяется знаком «=». Операции и константы записываются «как есть». Такая форма записи позволяет представлять инструкции в виде математических формул. Например, инструкции add eax,ecx и inc eax запишутся в виде:

$$P2 = P1 + P0 \quad (1)$$

$$P3 = P2 + 1 \quad (2)$$

Тем самым наглядно показывая процесс преобразования информации для участка машинного кода.

За этапом интерпретации, согласно предлагаемому алгоритму декомпиляции, следует этап сборки, идея такого процесса состоит в том, что ресурс — источник одной операции может выступать в роли приемника в другой операции, таким образом устанавливая связь между операциями на участке кода.

Развивая методику сборки, для установки связи между операциями было предложено использовать аппарат синтаксических деревьев для описания преобразований над ресурсами. В таком дереве узлы представляют собой операторы из базового набора операций, описанного выше, а листья соответствуют ресурсам или константам. Рассмотрим следующий пример для описания предложенного аппарата:

$$P2 = P0 + P1 \quad (3)$$

$$P4 = P2 * P3 \quad (4)$$

В результате итоговое выражение будет иметь вид:

P4 = (P0+P1)*P3 (5)

А синтаксическое дерево примет следующий вид (рис. 2).

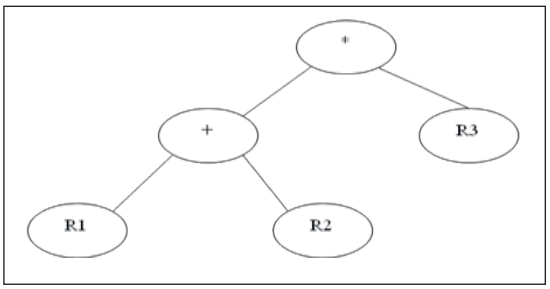

Рис. 2. Запись операций в виде бинарного дерева

Таким образом, для формирования итогового выражения необходимо осуществить рекурсивный проход по дереву вида: {левый лист, узел, правый лист}. Выход из узла дерева соответствует скобкам в итоговом выражении. Такие деревья необходимо формировать для каждого ресурса, являющегося выходным.

Для определения выходных ресурсов можно использовать несколько утверждений:

1) Выходным называется тот ресурс, который не является промежуточным.

2) В качестве выходных ресурсов можно рассматривать ресурсы памяти (в том числе стек программы), регистры общего назначения, так как чаще всего они являются выходными ресурсами при вызове функций.

В процессе полиморфной генерации бинарного кода возникает задача учета ресурсов процессора для обеспечения неизменности алгоритма работы участка кода, на котором производится генерация.

Для учета ресурсов процессора необходимо учитывать не только регистры и флаги процессора, но и регистры, которые используются для адресации памяти (например, mov [eax],3), так как их изменение, в некоторых случаях может привести к нарушению алгоритма работы участка кода.

Таким образом, для учета ресурсов процессора необходимо учитывать их изменения после каждой инструкции.

Для учета ресурсов процессора используется битовая маска регистров общего назначения, таким образом, для генерации кода возможно использовать только те регистры, чей флаг не взведен. Также стоит отметить, что регистры, чей флаг взведен, нельзя использовать в качестве приемника информации, в качестве источника информации для других регистров их использование оправдано, так как такая пересылка информации не вносит изменений в алгоритм работы участка кода.

Апробация методики

Апробация вышеописанной методики проводилась на двух следующих задачах:

- Защита участка программного кода;
- Автоматизированный анализ полиморфного кода.

А. Защита участка программного кода

На практике любая атака на программное обеспечение основывается на предварительном анализе защитного механизма приложения, зачастую с использованием дизассемблирования бинарного кода. Цель такого исследования заключается в том, чтобы восстановить алгоритм защиты, выделить его слабые стороны или недокументированные возможности для его последующей модификации и (или) автоматизации процесса преодоления. Такое преодоление гарантированно достигается за конечное время вследствие конечности программы, а время, за которое злоумыш-

ленник преодолеет систему, зависит от сложности этой системы и его квалификации. Способствует такой ситуации широкий выбор инструментов для исследования программного кода.

Технология полиморфной генерации кода позволяет проводить запутывающие преобразования в бинарном коде защищаемого объекта. В рамках данной работы производится встраивание (вставка запутывающих инструкций между инструкциями защищаемого кода) запутывающего кода в бинарный код защищаемого объекта. Такое встраивание позволяет увеличить время, за которое злоумышленник восстановит алгоритм защиты, за счет усложнения исследуемого кода:

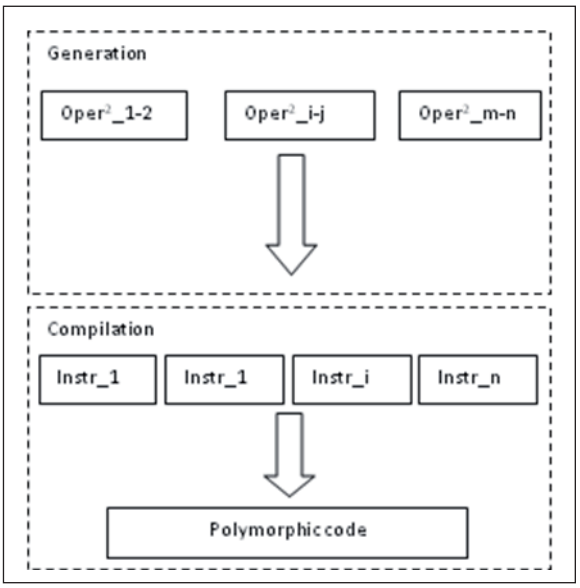

Рис. 3. Алгоритм генерации полиморфного кода

Таким образом, согласно рис. 3, для встраивания полиморфной инструкции необходимо еще несколько дополнительных этапов, таких как генерация инструкции и ее компиляция.

Стоит отметить, что реализация этапа интерпретации не требует анализа всего множества инструкций архитектуры x86. Этот факт обусловлен тем, что компиляторы используют не весь набор инструкций данной архитектуры. Было установлено в результате предварительного анализа частоты появления тех или иных инструкций в бинарном коде открытого программного обеспечения (рис. 4).

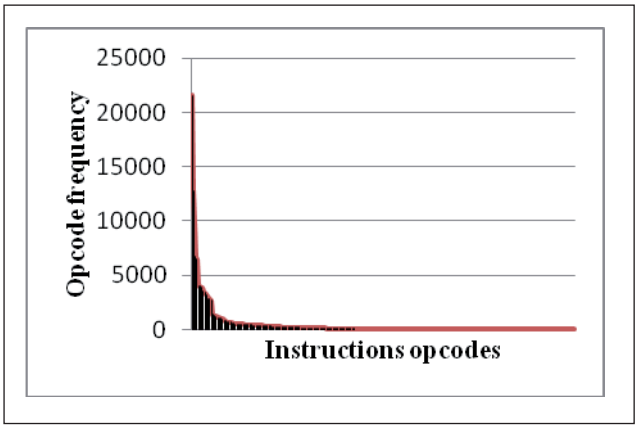

Рис. 4. Статистика x86 инструкций

Объем выборки составил 1086980 инструкций свободно распространяемого программного обеспечения.

Методика генерации кода (описанная ранее) была реализована в виде программного средства, которое включает в себя ряд базовых модулей, таких как анализатор кода и генератор полиморфного кода, а также базу данных, содержащую набор инструкций и соответствующий им алгоритм (рис. 5).

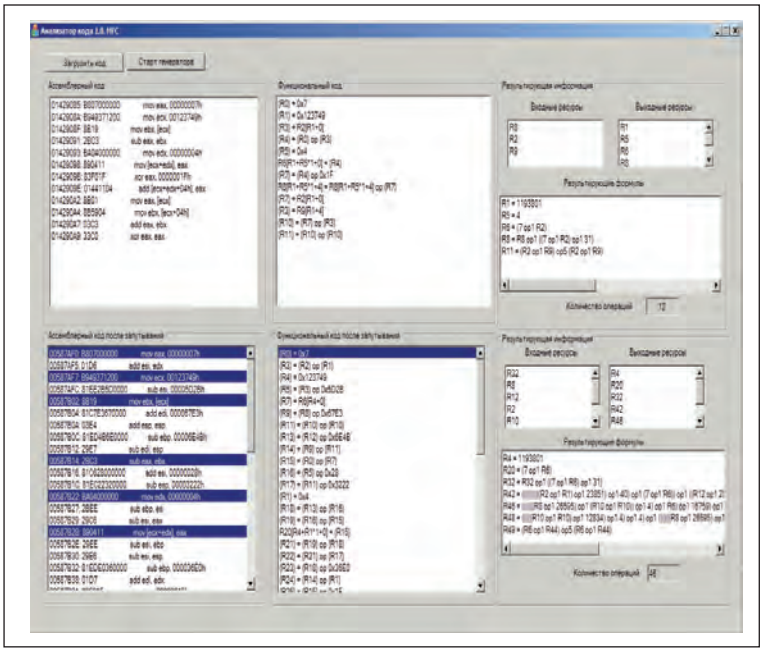

Рис. 5. Генератор полиморфного кода

Полученные результаты позволяют говорить об эффективности предложенной методики: количество операций на участке кода, количество входных и выходных ресурсов, а также результирующих формул возросло, что в свою очередь непременно приведет к росту ресурсов, требуемых для исследования данного участка кода, а это в свою очередь повышает защищенность описываемого участка от исследования.

B. Автоматизированный анализ полиморфного кода

Второй вариант апробации предложенной методики заключается в автоматизации анализа секций полиморфного кода вируса.

Полиморфный вирусный код — это код, который модифицирует сам себя (свое бинарное представление), при этом алгоритм, выполняемый участком кода, остается неизменным. Такой алгоритм представляет собой сигнатуру полиморфного вируса и может быть использован для обнаружения вируса.

Отметим, что для обнаружения полиморфного кода необходимо руководствоваться следующим утверждением: полиморфный код — это код, который не имеет выходных ресурсов и не участвует в их формировании.

Таким образом, с помощью вышеописанной методики декомпиляции бинарного кода, а также утверждения о полиморфном коде была реализована программа, позволяющая детектировать полиморфные инструкции вирусного кода в автоматизированном режиме (рис. 6).

Описанное средство позволяет детектировать полиморфные инструкции без ручного (с использованием средств дизассемблирования) анализа вирусного кода. Необходимо отметить, что на данный момент описанное детектирование возможно лишь на линейных участках полиморфного кода (участок кода без использования ветвления и вызова подпрограмм).

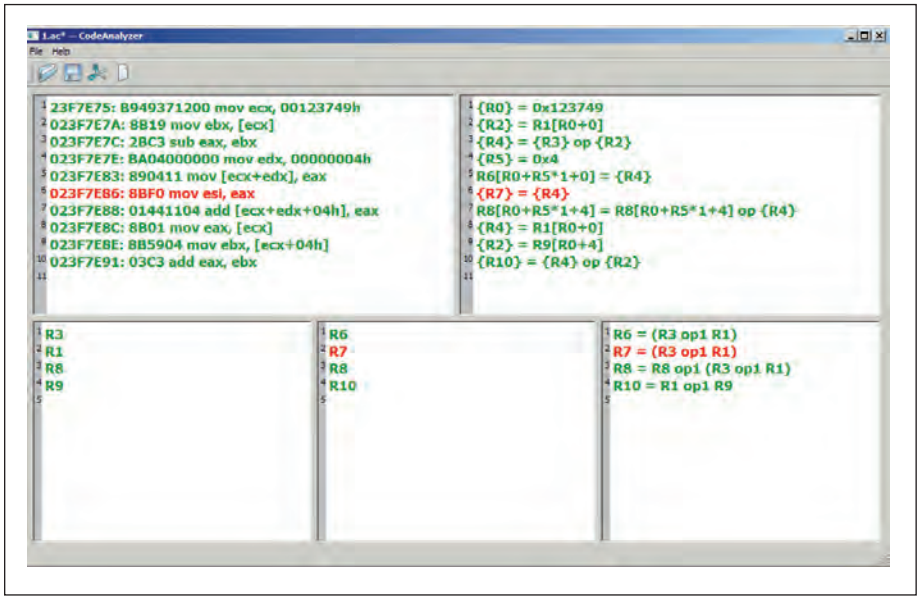

Рис. 6. Анализатор полиморфного кода вируса

Заключение

Таким образом, в данной статье была описана методика декомпиляции бинарного кода посредством применения оригинального алгоритма представления инструкции в алгоритмическом виде.

Описанная методика декомпиляции может найти широкое применение в различных аспектах информационной безопасности, особенно в разделах, касающихся защиты программного обеспечения и анализа вредоносного кода, что и было продемонстрировано авторами статьи. Исследование выполнено при поддержке Министерства образования и науки Российской Федерации, соглашение 14.132.21.1365 «Разработка программного комплекса декомпиляции бинарного кода»

СПИСОК ЛИТЕРАТУРЫ:

1. *Stiff G., Vahid F.* New decompilation technique for binary-level co-processor generation // Computer-Aided Design. 2005. ICCAD 2005. IEEE/ACM International. San Jose, CA, 2005.
2. *Cifuentes C., Waddington T., Van Emmerik M.* Computer security analysis through decompilation and high-level debugging // Reverse Engineering, 2001. Proceedings. Eighth Working Conference. Microsyst. Labs, Palo Alto, CA, 2001.