Поиск похожих последовательностей кода в исполняемых файлах на основе структурного анализа функций

А.С. Юмаганов[1], В.В. Мясников[1,2]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086
[2]Институт систем обработки изображений РАН – филиал ФНИЦ «Кристаллография и фотоника» РАН, Молодогвардейская 151, Самара, Россия, 443001

Аннотация. Статья посвящена разработке метода поиска похожих последовательностей кода в исполняемых файлах, основанного на анализе графов потока управления функций. Данный метод использует беспризнаковый подход, при котором описание функций формируется через отношение схожести с функциями наперед заданной базисной библиотеки. Представлены результаты экспериментальных исследований, демонстрирующие работоспособность разработанного метода и его эффективность в сравнении с ранее известным методом.

1. Введение

В настоящее время при разработке нового программного обеспечения (ПО) разработчики часто используют программные модули/библиотеки, реализованные ранее другими людьми. Например, согласно исследованиям, представленным в работе [1], около 10% файлов проектов с открытым исходным кодом на языке Java являются повторно использованными. Такой подход к разработке ПО может привести к переносу ошибок и уязвимостей в разрабатываемое ПО, а так же стать причиной нарушения лицензионного соглашения. Решение задачи поиска похожих последовательностей кода в исполняемых файлах можно использовать для нахождения такого рода ошибок и уязвимостей, выявления плагиата в программном коде. Кроме того, большая часть новых вредоносных программ является модификацией ранее известных, что позволяет использовать методы поиска похожих последовательностей кода для их детектирования.

Существует большое количество известных методов и алгоритмов поиска похожих последовательностей кода в исполняемых файлах. В работе [2] представлен метод поиска похожих последовательностей кода, основанный на сравнении последовательностей команд процессора (k-grams) внутри функций, полученных с помощью скользящего окна фиксированного размера (k). Метод поиска похожих последовательностей кода в работе [3] основан на сравнении статических переменных внутри исполняемого файла. Этот подход позволяет проводить анализ обфусцированных файлов, но обладает низкой точностью поиска. Названные выше методы основаны на синтаксическом анализе кода исполняемого файла. Такие методы очень чувствительны к различным изменениям непосредственно кода: замене команд процессора, перестановке команд, вставке новых команд, изменению статических переменных и т.д. Этого недостатка лишены методы, основанные на структурном анализе

функций. Например, в работах [4,5] описаны методы, в которых поиск вредоносных программ основан на определении изоморфности подграфов функций фиксированной длины и сравнении сигнатур («fingerprints») их базовых блоков. Эти методы в свою очередь сильно чувствительны к структурным изменениям кода.

Данная работа посвящена разработке метода поиска функций исполняемого файла, схожих с известными функциями из некоторого "архива". Описание функции в представленном методе формируется через ее отношения схожести с функциями, составляющими базисную библиотеку. Такой подход был использован в предыдущих работах авторов [6,7,8], где в основе сравнения функций лежал синтаксический анализ кода. В основе же представленного в данной работе метода лежит анализ структуры графов потока управления функций.

Работа построена следующим образом. Во втором разделе представлено краткое описание разработанного метода. В третьем разделе описан процесс получения первичного описания функций исполняемого файла на основе анализа их графов потока управления. В четвертом разделе рассматривается процесс получения промежуточного и окончательного представления функции через библиотеку базисных функций. В пятом разделе представлены результаты экспериментальных исследований. В заключении приводятся выводы, представлен список использованной литературы.

2. Основные понятия и принцип работы

Введем следующие понятия:

- архивная библиотека – набор известных функций и их описание через библиотеку базисных функций.
- текущая библиотека – набор функций исследуемого исполняемого файла и их описание через библиотеку базисных функций.
- библиотека базисных функций – вспомогательный набор функций, используемое для сравнения функций архивной и текущей библиотек.

Решаемая разработанным методом задача формулируется следующим образом: для заданной функции текущей библиотеки найти среди функций архивных данных наиболее похожую функцию. Определение меры сходства двух функций может быть задано множеством способов. В предыдущей работе авторов [8], например, сходство функций определялось на основе расположения команд процессора различных функциональных групп в теле рассматриваемых функций. В рамках данной работы используется мера сходства, основанная на сравнении подграфов графа потока управления функций. Подробное описание используемой меры сходства представлено в 4 разделе.

Представленный в данной работе метод поиска похожих последовательностей кода включает в себя три этапа. На первом этапе происходит представление функций архивных данных через отношение схожести с функциями базисной библиотеки, полученное описание функций сохраняется в базу данных (БД) архивных функций . На втором этапе аналогичным образом формируется представление функций текущей библиотеки, которое так же сохраняется в соответствующей БД. На заключительном этапе осуществляется непосредственно поиск похожих функций. Первый и второй этап для фиксированных архивной и текущей библиотек выполняются только один раз, третий этап (поиск) в данном случае может выполняться необходимое число раз.

3. Анализ графа потока управления функции и получение ее первичного описания

Используя дизассемблер IDA[9], можно провести анализ исследуемого исполняемого файла. В результате которого, получим ассемблерный код данного файла с разбиением на функции и графы потока управления для каждой их функций.

Граф потока управления функции (control flow graph) – это ориентированный граф, вершинами которого являются базовые блоки функции (basic blocks), а ребра определяют порядок следования базовых блоков в потоке управления функции. Базовый блок представляет

собой последовательность команд процессора, не включающую в себя команд передачи управления (за исключением последней команды базового блока и команды вызова функций "call") и команд на которые управление передается (исключение - первая команда базового блока).

Из полученного графа потока управления для заданной функции формируется набор подграфов фиксированного порядка k (k-подграфы). Этот набор получается путем поочередного обхода графа потока управления функции, начиная с каждого ее базового блока, в глубину до тех пор, пока не будет пройдено k вершин. Для полученных таким образом k-подграфов получаем матрицу смежности размерности $k*k$.

Таким образом, структура каждого k-подграфа может быть представлена в виде бинарного вектора $\bar{a}$ размерности $k*k$, который получается путем объединения строк его матрицы смежности. Значение величины k является одним из параметров метода. Схожим образом структурное описания подграфов было получено в работе [10].

Для увеличения точности поиска помимо информации о структуре подграфов необходимо учитывать и содержание их базовых блоков. Каждый k-подграф функции будем описывать бинарным вектором $\bar{b}$, характеризующим наличие или отсутствие в нем операций, осуществляющих чтение/запись данных в операнды различных типов. В таблице 1 представлено подробное описание элементов данного вектора. При наличии i-ой характеристики в рассматриваемом подграфе значение i-ого бита вектора $\bar{b}$ равно 1, при отсутствии – 0. Таким образом, два k-подграфа эквивалентны друг другу, если они имеют одинаковую структуру (равные вектора $\bar{a}$) и соответствующие им вектора $\bar{b}$ равны.

Таблица 1. Описание вектора $\bar{b}$.

№ элемента вектора $\bar{b}$	Описываемая характеристика
1	Чтение данных из регистров
2	Запись данных в регистры
3	Чтение данных по прямой ссылке на ячейку памяти
4	Запись данных по прямой ссылке на ячейку памяти
5	Чтение данных по косвенной ссылке на ячейку памяти
6	Запись данных по косвенной ссылке на ячейку памяти
7	Наличие операнда, представляющего собой непосредственно значения
8	Наличие прямой ссылки на некоторый участок кода.

Для каждого k-подграфа исследуемой функции получаем пару векторов $\bar{a}$ и $\bar{b}$. Набор пар таких векторов для заданной функции формирует ее первичное описание. Полученное таким образом описание функции заносится в соответствующую БД и в дальнейшем используется для построения промежуточного описания функции с помощью библиотеки базисных функций.

4. Формирование промежуточного и окончательного описания функций

Для формирования промежуточного описания исследуемой функции необходимо получить значение некоторой меры схожести с каждой из функций базисной библиотеки. В качестве такой меры схожести воспользуемся обобщенным коэффициентом Жаккара (generalized Jaccard index) [11]:

$$J(x,y)=\frac{\sum_i \min(x_i,y_i)}{\sum_i \max(x_i,y_i)}, \quad (1)$$

где x – набор пар векторов, описывающий первую функцию, y – набор пар векторов, описывающий вторую функцию, x_i – количество пар i в наборе x, y_i – количество пар i в

наборе *y*, *i* проходит по всем уникальным парам векторов в объединенном наборе $x \cup y$. При полном совпадении двух функций мера схожести (1) принимает значение «1», при полном несоответствии – «0».

Пусть *N* – число функций в базисной библиотеке, каждая из которых имеет описание в виде набора пар векторов y_n, x – набор пар векторов, описывающий исследуемую функцию. Сравнивая рассматриваемую функцию с каждой из функций базисной библиотеки с помощью формулы (1) , получим следующий вектор промежуточного описания:

$\bar{z} = (J(x, y_0), J(x, y_1), ..., J(x, y_{N-1}))^T$.

Так как $N >> 1$, воспользуемся методом снижения размерности – методом главных компонент (англ. Principal component analysis, PCA). В результате применения метода PCA получим новый вектор $\bar{f}$ размерности $I < N$:

$\bar{f} = Y\bar{z}$,

где *Y* – матрица перехода к новой размерности, составленная из *I* собственных векторов.

Полученный вектор $\bar{f}$ выступает в качестве окончательного представления исследуемой функции через библиотеку базисных функций и заносится в соответствующую БД (архивную или текущую библиотеку).

5. Результаты экспериментов

Для оценки эффективности представленного метода поиска похожих последовательностей кода будем использовать в качестве библиотеки архивных данных функции некоторой динамической библиотеки (libtiff[12], libcurl[13]), а в качестве текущей - функции той же самой библиотеки другой версии. Для определения априори похожих функций считалось, что при переходе от одной версии динамической библиотеки к другой имена функций не меняются.

Для исследуемой функции текущей библиотеки получим список функций архивных данных, отсортированный по уменьшению схожести, с помощью алгоритма, представленного в предыдущей работе авторов [6]. Поставим в соответствие этому списку бинарную последовательность $\beta = (\beta_1, \beta_2, ..., \beta_L)$, *i*-ый элемент которой равен единице, если имя функции на *i*-ой позиции списка идентично имени проверяемой функции, и равен нулю в противном случае. Тогда воспользуемся критериями оценки качества информационного поиска[14][15]:

- Точность для k-ой позиции списка *:* $P_k = \frac{\sum_{l=1}^{k} \beta_l}{k}$
- Полнота для k-ой позиции списка *:* $R_k = \frac{\sum_{l=1}^{k} \beta_l}{K}$
- Средняя точность для списка*:* $AveP = \sum_{k=1}^{L} P_k (R_k - R_{k-1}), R_0 = 0$.

Тогда средняя точность для всех функций входящих в состав текущей библиотеки вычисляется по формуле:

$$P = \frac{1}{S} \sum_{s=0}^{S-1} AveP_s , \quad (2)$$

где *S* – количество функций в текущей библиотеке.

При проведении экспериментальных исследований архивные данные будут представлены библиотеками libtiff 4.0.3 и libcurl 7.53.1, а функции текущей библиотеки - либо функциями библиотек libtiff различных версий (3.9.2, 4.0.8), либо соответственно функциями библиотек

libcurl (7.49.0, 7.51.0). Библиотека базисных функций состоит из N=50 различных функций, выбранных вручную из других исполняемых файлов и библиотек.

Установим минимальное количество базовых блоков в графе потока управления функций архивных данных и текущей библиотеки $bb_{\min} = 20$, чтобы исключить "маленькие" функции, первичное описание которых в силу их размера может быть сильно схожим с первичным описанием других таких функций, что может привести к снижению точности поиска.

Сравним значения средней точности поиска (2) для указанных выше библиотек при использовании различных значений параметра метода k (порядок подграфов). В таблице 2 представлены полученные результаты.

Таблица 2. Зависимость средней точности поиска от параметра k.

Архивные данные	Текущая библиотека	Средняя точность поиска P		
		k=3	k=4	k=5
libtiff 4.0.3	libtiff 3.9.2	**0.8313**	0.8143	0.7862
libtiff 4.0.3	libtiff 4.0.8	**0.9117**	0.8966	0.8674
libcurl 7.53.1	libcurl 7.49.0	0.9281	**0.9308**	0.8766
libcurl 7.53.1	libcurl 7.51.0	**0.9579**	0.9576	0.9078

При значении параметра k=3 средняя точность поиска в большинстве случаев выше. Это объясняется тем, что при большем значении k вектор $\bar{b}$, описывающий содержание базовых блоков k-подграфа, характеризует большее число базовых блоков. Вероятность наличия каждой из описываемых вектором $\bar{b}$ характеристик подграфа увеличивается, что снижает его уникальность для каждого подграфа. В результате средняя точность поиска снижается.

Сравним эффективность представленного в данной работе метода поиска похожих последовательностей кода с представленным в предыдущей работе авторов методом [8]. Как было отмечено выше, представленный в работе [8] метод основан на сравнении пространственного положения команд процессора в теле функции. В качестве объекта сравнения (параметр метода) будем использовать рекомендуемый авторами объект – пространственное распределение команд в теле функции в интегральной форме. Для функций текущей и архивной библиотек, используемых этим методом, установим аналогичное ограничение на минимальное количество базовых блоков в функции $bb_{\min} = 20$. Полученные результаты представлены в таблице 3.

Таблица 3. Сравнение методов поиска похожих функций ($bb_{\min} = 20$).

Архивные данные	Текущая библиотека	Средняя точность поиска P, используя разработанный метод (k = 3)	Средняя точность поиска P, используя представленный в работе [8] метод
libtiff 4.0.3	libtiff 3.9.2	**0.8313**	0.8039
libtiff 4.0.3	libtiff 4.0.8	0.9117	**0.9286**
libcurl 7.53.1	libcurl 7.49.0	**0.9281**	0.8950
libcurl 7.53.1	libcurl 7.51.0	**0.9579**	0.9342

Анализ полученных результатов показывает, что представленный в данной работе метод поиска похожих последовательностей кода в большинстве случаев превосходит метод, представленный в [8] по критерию средней точности поиска при заданном ограничении $bb_{\min}$. Однако, следует отметить, что при уменьшении величины $bb_{\min}$ до 5 средняя точность поиска оригинального метода существенно снижается и лучшие результаты показывает описанный в работе [8] метод. В таблице 4 представлены результаты сравнения двух методов при $bb_{\min} = 5$.

Таблица 4. Сравнение методов поиска похожих функций ($bb_{\min} = 5$).

Архивные данные	Текущая библиотека	Средняя точность поиска P, используя разработанный метод (k = 3)	Средняя точность поиска P, используя представленный в работе [8] метод
libtiff 4.0.3	libtiff 3.9.2	0.7990	**0.8218**
libtiff 4.0.3	libtiff 4.0.8	0.8446	**0.8995**
libcurl 7.53.1	libcurl 7.49.0	0.8973	**0.9116**
libcurl 7.53.1	libcurl 7.51.0	0.9320	**0.9453**

Все представленные выше результаты экспериментальных исследований были получены при использовании библиотек, скомпилированных с флагом оптимизации /Od (оптимизация отключена). Ассемблерный код этих библиотек существенно отличается от кода соответствующих версий тех же библиотек, скомпилированных с включенной оптимизацией (флаги /O1, /O2, /Ox). Как показали экспериментальные исследования, если архивные данные представлены библиотекой, скомпилированной с отключенной оптимизацией (/Od), а текущая библиотека - библиотекой, скомпилированной с включенной оптимизацией, то найти похожие функции, используя представленный в данной работе метод, не удастся. Однако, функции библиотеки, скомпилированной с флагом /O1, можно использовать для нахождения похожих функций в библиотеке, скомпилированной с флагом оптимизации /Ox. Тем не менее при наличии исходного кода программы ее можно скомпилировать несколько раз со всеми возможными вариантами настройки оптимизации и занести функции каждого полученного экземпляра этой программы в архивную библиотеку. Таким образом, похожие функции могут быть найдены независимо от настроек оптимизации компилятора функций текущей библиотеки.

6. Заключение

В работе представлен метод поиска похожих последовательностей кода в исполняемых файлах, в котором описание функций формируется на основе сравнения подграфов фиксированного порядка графов потока управления этих функций и функций, составляющих базисную библиотеку. Приводятся результаты экспериментальных исследований, демонстрирующие работоспособность разработанного метода и его превосходство над другим ранее известным методом. Дальнейшие исследования будут направлены на улучшение эффективности представленного метода при его работе с функциями, граф потока управления которых состоит из небольшого числа базовых блоков.

7. Благодарности

Теоретическая часть работы (разделы 2-4) выполнена при финансовой поддержке РФФИ в рамках научного проекта № 18-31-00350-мол_а. Экспериментальная часть работы (раздел 5) выполнена в рамках госзадания ФАНО России, проект № 0026-2018-0106.

8. Литература

[1] Ossher, J. File cloning in open source Java projects: the good, the bad, and the ugly / J. Ossher, H. Sajnani, C. Lopes // Proceedings of the International Conference in Software Maintenance, Williamsburg, VI, 2011. – P. 283-292.

[2] Myles, G. K-gram based software birthmarks / G. Myles, C. Collberg // Proceedings of the 2005 ACM symposium on Applied computing, ser. SAC ’05. – ACM, 2005. – P. 314-318.

[3] Hemel, A. Dolstra Finding software license violations through binary code clone detection / A. Hemel, K.T. Kalleberg, R.E. Vermaas // Proceedings of the 8th International Working Conference on Mining Software Repositories (MSR 2011), 2011. – P. 63-72.

[4] Flake, H. Structural comparison of executable objects / H. Flake // Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment, 2004. – P. 161-173.
[5] Kruegel C, Kirda E, Mutz D, Robertson W, Vigna G. Polymorphic worm detection using structural information of executables. RAID'05 2005: 207-226. DOI: 10.1007/11663812_11.
[6] Yumaganov AS, Myasnikov VV. Similarity search over program code sequences using featureless pattern recognition techniques. CEUR Workshop Proceedings, 2016; 1638: 437-443. DOI: 10.18287/1613-0073-2016-1638-437-443
[7] A.S. Yumaganov, V.V. Myasnikov, Comparison of the ways of the program's code initial description in the problem of similar code sequences search, In Procedia Engineering, Volume 201, 2017, Pages 445-452, ISSN 1877-7058
[8] Yumaganov AS, Myasnikov VV. A method of searching for similar code sequences in executable binary files using a featureless approach. Computer Optics 2017; 41(5): 756-764. DOI: 10.18287/2412-6179-2017-41-5-756-764.
[9] Hex-Rays IDA: About [Electronic resource]. — Access mode: http://hex-rays.com/products/ida/ (09.11.2017).
[10] W. M. Khoo, A. Mycroft, and R. J. Anderson. Rendezvous: a search engine for binary code. In Proc. of MSR'13, 2013.
[11] H. Spath. The minisum location problem for the Jaccard metric. Operations Research-Spektrum 3, 91–94, 1981.
[12] TIFF Library and Utilities [Electronic resource]. — Access mode: http://www.libtiff.org/ (09.11.2017).
[13] Curl: command line tool and library [Electronic resource]. — Access mode: http://www.curl.haxx.se/ (09.11.2017).
[14] Buckland, M. K. The relationship between recall and precision/ M.K. Buckland, F.C. Gey // JASIS. – 1994. – Vol. 45. No. 1. – P. 12-19.
[15] Powers, D. M. W. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. // Journal of Machine Learning Technologies. – 2011. – Vol. 2. No. 1. – P. 37-63.

Searching for similar code sequences in executable files based on the structural analysis of functions

A.S. Yumaganov[1], V.V. Myasnikov[1,2]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086
[2]Image Processing Systems Institute of RAS - Branch of the FSRC "Crystallography and Photonics" RAS, Molodogvardejskaya street 151, Samara, Russia, 443001

Abstract. This work is devoted to the development of a method of similar code sequences search in executable files, based functions control flow graph analysis. This method is also based on featureless approach, where functions are represented by their relationships with predefined basis library functions. The results of experimental studies demonstrates applicability of proposed method and its efficiency in comparison with another method of similar code sequences search.

Keywords: control flow graph, searching, code sequences, featureless recognition.