Комбинированный метод поиска похожих последовательностей кода в исполняемых файлах

А.С. Юмаганов[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. Статья посвящена разработке метода поиска похожих последовательностей кода в исполняемых файлах, в основе которого лежит синтаксический анализ кода и анализ графов потока управления функций. Используемый в данной работе метод синтаксического анализа кода основан на сравнении пространственного расположения команд процессора в теле функции. Для анализа графа потока управления функции используется структурное описание его подграфов фиксированного порядка. Представлены результаты экспериментальных исследований эффективности предложенного метода по сравнению с известными ранее методами поиска похожих последовательностей кода.

1. Введение

Проблема поиска похожих последовательностей кода в исполняемых файлах в настоящее время является очень актуальной. Как показали исследования, представленные в [1,2], при разработке нового программного обеспечения (ПО) разработчики часто используют ранее созданный программный код. Такой подход к разработке ПО называется повторное использование кода (code reuse). Несмотря на очевидные преимущества данного подхода, он так же может стать причиной появления ошибок и уязвимостей в разрабатываемом ПО. Кроме того, повторное использование кода сторонних разработчиков может быть незаконным. Данный подход так же используется при разработке различных вредоносных программ [3]. Таким образом, решение задачи поиска похожих последовательностей кода в исполняемых файлах позволяет решить целый ряд проблем: поиск известных уязвимостей, нахождение плагиата в ПО, поиск вредоносных программ.

В настоящее время известно большое количество методов поиска похожих последовательностей кода в исполняемых файлах. Все известные алгоритмы и методы решения указанной выше задачи обычно основываются либо на синтаксическом анализе ассемблерного кода программ, либо на анализе структуры графов потока управления (control flow graph) функций исполняемого файла.

Рассмотрим методы, в основе которых лежит синтаксический анализ кода. В работах [4,5] представлены схожие методы поиска похожих последовательностей кода, основанные на сравнении последовательностей команд процессора фиксированной длины (k-grams) внутри функций или перестановок команд процессора (n-perms) соответственно. В дизассемблере IDA для идентификации библиотечных функций используется алгоритм IDA FLIRT [6], основанный на сравнении «шаблонов» функций. Автор [7] представил метод детектирования вредоносных

программ, основанный на анализе частоты встречаемости команд процессора внутри исследуемого файла. Существенное влияние на качество поиска похожих последовательностей кода данной группы методов оказывают синтаксические изменения кода: замена команд процессора на эквивалентные, перестановка команд, вставка новых и удаление старых команд.
Использование методов, в основе которых лежит анализ графа потока управления функции, позволяет преодолеть этот недостаток. В работе [8] для детектирования и классификации вредоносных программ используется информация о базовых блоках, которые являются вершинами графа потока управления. В [9] для выявления вредоносных программ авторы используют метод, основанный на определении изоморфности графов потока управления. Схожий подход описан в работе [10], где поиск вредоносных программ основан на определении изоморфности подграфов функций фиксированной длины и сравнении сигнатур («fingerprints») их базовых блоков. Однако данная группа методов так же имеет ряд недостатков: низкая точность поиска при использовании данных методов для функций с малым числом базовых блоков, высокая чувствительность к структурным изменениям функций.
В данной работе предлагается комбинированный метод поиска функций исполняемого файла, схожих с известными функциями из некоторого "архива" программ. В основе предлагаемого метода лежит применение, как синтаксического анализа кода, так и анализа графа потока управления функций. Описание функции в представленном методе формируется через ее отношения схожести с функциями, составляющими базисную библиотеку.
Работа построена следующим образом. В первом разделе представлены основные определения и краткое описание предложенного метода. Во втором разделе рассматривается процесс получения синтаксического, в третьем - структурного описания функций. Четвертый раздел посвящен описанию алгоритма поиска похожих функций. В пятом разделе приводится способ оценки эффективности метода и результаты проведенных экспериментов. В заключении приводятся выводы, представлен список использованной литературы.

2. Основные понятия и принцип работы

В данной работе используются следующие определения:

- текущая библиотека – множество функций исследуемого исполняемого файла;
- архивные данные – множество известных функций;
- базисная библиотека – вспомогательное множество функций, применяемое для сравнения функций архивных данных и текущей библиотеки.

С учетом представленных выше определений решаемая задача формулируется следующим образом: для заданной (или каждой) функции текущей библиотеки найти наиболее похожую функцию из архивных данных. В данной работе используются два определения меры сходства функций: на основе положения функциональных групп команд процессора в теле функций и на основе анализа структуры графа потока управления функций. В ранее опубликованных работах автора [11,12] были представлены два метода поиска похожих последовательностей кода, в основе каждого из которых лежит соответственно одно из названных выше определений меры схожести функций. В данной работе представлен комбинированный метод поиска похожих последовательностей кода, в котором описание функций формируется через отношение схожести с функциями базисной библиотеки, используя два различных определения меры схожести функций, описанных в работах [11,12].
Предлагаемый в данной работе метод поиска похожих последовательностей кода включает в себя несколько этапов. На первом этапе формируется описание функций архивных данных, через библиотеку базисных функций. При этом для каждой функции получается два описания (синтаксическое и структурное), соответствующих разным используемым мерам сходства функций. На втором этапе аналогичным образом формируется описание функций текущей библиотеки. На заключительном третьем этапе осуществляется непосредственно поиск функций, алгоритм которого подробно рассмотрен в пятом разделе.

3. Получение синтаксического описания функции

Кратко рассмотрим процесс получения синтаксического описания функции некоторого исполняемого файла. С помощью дизассемблера IDA [13] можно получить разбиение ассемблерного кода рассматриваемого исполняемого файла на функции. Ассемблерный код состоит из последовательности команд процессора и связанных с ними операндов. Все команды процессора можно разделить на *K* функциональных групп по типу выполняемых ими операций. Примерами таких групп являются: группа арифметических команд, группа логических команд, группа команд пересылки данных. Для каждой из K функциональных групп для заданной функции получим список смещений относительно начала функции, на котором располагаются команды этой группы.

Определим пространственное распределение k-го типа команд как абсолютную частоту попадания команд этого типа в некоторый (относительный) приведенный *i*-ый интервал $(I = 100)$:

$$\tilde{f}_i^k = \sum_{j=0}^{N_k - 1} I\left(\frac{n_j^k}{N} \cdot 100 \in (i-1, i] \right), \quad i = \overline{1, I} \tag{1}$$

где $n_0^k, ..., n_{N_k - 1}^k$ – абсолютные смещения (позиции) относительно начала функции команд группы k, N_k – общее количество команд этой группы в данной функции, *N* – длина функции, I(.) - индикатор события, принимающий значения "0" или "1" в зависимости от истинности соответствующего аргумента.

Для получения пространственного распределения команд в интегральной форме воспользуемся следующей формулой:

$$\hat{f}_i^k = \frac{\sum_{y=0}^{i} \tilde{f}_y^k}{\sum_{j=0}^{I} \tilde{f}_j^k}, i = \overline{1, I} \tag{2}$$

Тогда, пространственное положение k-ой группы команд процессора в теле рассматриваемой функции описывается вектором:

$$\overline{a}_k = \left(\hat{f}_1^k, \hat{f}_2^k, \ldots, \hat{f}_I^k \right)^T, k = \overline{0, K-1}. \tag{3}$$

В результате описание рассматриваемой функции имеет следующий вид:

$$A = (\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_{K-1}) \tag{4}$$

Аналогичным образом формируется матрица B описания функций базисной библиотеки. Тогда, мера схожести функций, описание которых задано матрицами A и B имеет следующий вид:

$$\mu(A, B) = \sum_{k=0}^{K-1} \alpha_k \mu_{\cos}(\overline{a}_k, \overline{b}_k), \quad \sum_{k=0}^{K-1} \alpha_k = 1, \tag{5}$$

где $\mu_{\cos}$ - косинусное расстояние. При полном совпадении функций мера схожести принимает значение "1", при полном несоответствии – "0".

Пусть библиотека базисных функций содержит J функций, каждая из которых имеет описание в виде матрицы B_j. Тогда описание рассматриваемой функции через библиотеку базисных функций будет иметь следующий вид:

$$\overline{x}_A = (\mu(A, C_0), \mu(A, C_1), \ldots, \mu(A, C_{J-1}))^T \tag{6}$$

Далее, используя полученное промежуточное описание функции, формируются ее окончательное описание с помощью метода снижения размерности данных PCA (principal component analysis). Процесс формирования окончательного описания подробно описан в [11]. Полученное описание функции заносится в соответствующую базу данных (архивную или текущую).

4. Получение структурного описания функции

Дисассемблер IDA позволяет получить граф потока управления для функций анализируемого исполняемого файла. Граф потока управления (control flow graph) функции – это ориентированный граф, вершинами которого являются базовые блоки функций. Базовый блок (basic block) функции – это последовательность команд процессора, на первую команду которой управление передается, а последняя ее команда – команда передачи управления. Ребра графа потока управления определяют порядок следования базовых блоков в потоке управления функции.

Граф потока управления исследуемой функции разбивается на подграфы фиксированного порядка *k* (*k*-подграфы) следующим образом: выполняется поочередный обход графа в глубину, начиная с каждого из базовых блоков данного графа, до тех пор, пока не будет пройдено *k* вершин. В данной работе используется *k*=3.

Описание каждого из *k*-подграфов исследуемой функции состоит из пары векторов: $\bar{a}$ и $\bar{b}$. Вектор $\bar{a}$ представляет собой бинарный вектор, полученный путем объединения строк матрицы смежности данного k-подграфа. Вектор $\bar{b}$ характеризует наличие или отсутствие в данном k-подграфе операций чтения или записи в операнды различного типа. Подробное описание вектора $\bar{b}$ представлено в [12]. Таким образом, первичное описание функции состоит из множества пар векторов $\bar{a}$ и $\bar{b}$, описывающих каждый k-подграф этой функции.

Промежуточное описание исследуемой функции формируется на основе ее схожести с функциями базовой библиотеки. В качестве меры схожести используем обобщенный коэффициент Жаккара (generalized Jaccard index) [14]:

$$J(x,y)=\frac{\sum_i \min(x_i,y_i)}{\sum_i \max(x_i,y_i)}, \quad (7)$$

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 $x \bigcup y$. При полном совпадении двух функций мера схожести (1) принимает значение «1», при полном несоответствии – «0».

Пусть x – набор пар векторов, описывающих исследуемую функцию; y_i - набор пар векторов, описывающих i-ую функцию базисной библиотек; I – количество функций в базисной библиотеке, тогда промежуточной описание исследуемой функции имеет следующий вид:

$$\bar{z}=(J(x,y_0),J(x,y_1),...,J(x,y_{I-1}))^T. \quad (8)$$

Окончательное описание функции получается после применение метода снижения размерности PCA и заносится в соответствующую базу данных (архивную или текущую).

5. Поиск похожих функций

Заключительным этап представленного в данной работе метода является непосредственно поиск похожих функций на основе полученных ранее векторов описания функций.

В предыдущих работах автора был использован алгоритм поиска, при котором для увеличения качества поиска использовалось следующее допущение: размер измененной функций отличается от размера оригинальной не более чем на 30% [11]. Таким образом, на первом этапе поиска осуществлялась фильтрация архивных функций по их размеру, затем вычислялось евклидово расстояние до каждой функции из отфильтрованного списка архивных функций и сортировка полученного результата по увеличению евклидова расстояния.

В данной работе представлен комбинированный метод поиска похожих функций, который использует синтаксическое и структурное описание функций. Однако, в том случае если количество базовых блоков исследуемой функции мало ($bb_{\min} \le 5$) используется только синтаксическое описание функции и описанный выше алгоритм поиска, применяемый автором

ранее. Данное условие позволяет повысить точность поиска, так как структурное описание малых функций может быть сильно схожим со структурным описанием таких же малых функций в силу их небольшого размера.
Представленный в данной работе метод поиска использует следующий алгоритм поиска похожих функций (при условии $bb_{\min} > 5$):

- На первом этапе осуществляется предварительная фильтрация архивных функций. Для исследуемой функции и всех функций архивных данных находится евклидово расстояние между векторами синтаксического (или структурного) описания функций и сортируется по увеличению расстояния. Первые $top_{th} = 30$ элементов полученного списка наиболее похожих архивных функций используются на втором этапе поиска.
- На втором этапе поиска вычисляется евклидово расстояние от исследуемой функции до полученного выше списка функций. Однако в данном случае в качестве векторов описания используются векторы отличные от тех, что использовались на первом этапе. Другими словами, если на первом этапе сравнение функций осуществлялось по векторам соответствующих их синтаксическому описанию, то на втором этапе - по векторам соответствующих их структурному описанию и наоборот. Затем осуществляется сортировка полученного результата по увеличению евклидового расстояния.

В результате для исследуемой функции получается список функций архивных данных, упорядоченный по уменьшению схожести.

6. Результаты экспериментов

Для оценки эффективности представленного метода поиска похожих последовательностей кода в исполняемых файлах будем использовать функции одной динамической библиотеки в качестве архивных данных, а функции такой же библиотеки, но другой версии – в качестве текущей библиотеки. Для определения априори похожих функций считалось, что при переходе от одной версии динамической библиотеки к другой имена функций не менялись и среди функций архивных данных отсутствуют функции с одинаковыми именами.
Используя описанный в пятом разделе алгоритм поиска, для заданной функции текущей библиотеки получим упорядоченный по уменьшению схожести список функций архивных данных. 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[15,16]:

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

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

$$P = \frac{1}{S} \sum_{s=0}^{S-1} AveP_s , \tag{9}$$

где *S* – количество функций в текущей библиотеке.

При проведении экспериментальных исследований были использованы несколько версий библиотеки libtiff [17]. В качестве функций архивных данных использовались функции библиотеки libtiff 4.0.8. Данные библиотеки были скомпилированы с флагом оптимизации /Od (оптимизация отключена).
Сравним значения средней точности поиска (9) при использовании двух способов предварительной фильтрации функций архивных данных: по синтаксическому описанию функций или по структурному описанию. Полученные результаты представлены в таблице 1.

Таблица 1. Сравнение способов предварительной фильтрации архивных функций.

Текущая библиотека	Средняя точность поиска P, фильтрация по синтаксическому описанию	Средняя точность поиска P, фильтрация по структурному описанию
libtiff 3.9.2	0.7424	**0.7592**
libtiff 3.9.7	0.7551	**0.7707**
libtiff 4.0.3	0.7835	**0.8290**
libtiff 4.0.5	0.7943	**0.8432**

При предварительной фильтрации функций архивных данных по структурному описанию средняя точность поиска для каждой из рассмотренных библиотек выше. Поэтому в дальнейших экспериментах будем использовать данный способ предварительной фильтрации архивных функций.
Проведем сравнение средней точности поиска представленного в данной работе метода поиска и известных ранее методов: метод, основанный на анализе пространственного положения функциональных групп команд процессора [11] и метод, основанный на сравнении k-грамм [4]. В качестве объекта сравнения для первого метода использовался рекомендуемый авторами объект сравнения - пространственное распределение команд в теле функции в интегральной форме. Для второго метода значение параметра k=5 было так же выбрано исходя из рекомендаций авторов. Полученные результаты представлены в таблице 2.

Таблица 2. Сравнение методов поиска похожих функций.

Текущая библиотека	Средняя точность поиска P, используя разработанный метод	Средняя точность поиска P похожих функций, используя метод, основанный на сравнении k-грамм функции	Средняя точность поиска P, используя представленный в работе [11] метод
libtiff 3.9.2	**0.7592**	0.7528	0.7370
libtiff 3.9.7	**0.7707**	0.7681	0.7524
libtiff 4.0.3	**0.8290**	0.7970	0.8257
libtiff 4.0.5	**0.8432**	0.8119	0.8427

Анализ полученных результатов показывает, что представленный в данной работе метод поиска похожих последовательностей кода превосходит известные ранее методы на каждой из используемых текущих библиотек. При этом чем «ближе» версия текущей библиотеки к версии библиотеки архивных данных, тем меньшее преимущество имеет представленный метод над методом [11]. Это объясняется тем, что для используемых при экспериментальных исследованиях библиотек некоторые функции более старых версий текущей библиотеки относительно архивной имеют сильные синтаксические изменения. И предварительная фильтрация архивных данных по структурному описанию позволяет значительно повысить точность поиска.

7. Заключение

В работе представлен комбинированный метод поиска похожих последовательностей кода в исполняемых файлах, использующий как синтаксическое, так и структурное описание функций. Представлены результаты экспериментов, демонстрирующих превосходство разработанного метода над некоторыми ранее известными методами. Дальнейшие исследования будут направлены на улучшение точности поиска похожих функций и исследование работоспособности и эффективности представленного метода при анализе исполняемых файлов, скомпилированных при различных настройках компиляции.

8. Литература

[1] Abdalkareem, R. On code reuse from StackOverflow: An exploratory study on Android apps / R. Abdalkareem, E. Shihab, J. Rilling // Information and Software Technology. – 2017. – Vol. 88. – P. 148-158.

[2] Gharehyazie, M. Some from here, some from there: cross-project code reuse in GitHub / M. Gharehyazie, B. Ray, V. Filkov // Proceedings of the 14th International Conference on Mining Software Repositories. – 2017. – P. 291-301.

[3] Examining Code Reuse Reveals Undiscovered Links Among North Korea's Malware Families [Electronic resource]. – Access mode: https://securingtomorrow.mcafee.com/mcafee-labs/examining-code-reuse-reveals-undiscovered-links-among-north-koreas-malware-families/ (05.11.2018).

[4] Myles, G. K-gram based software birthmarks /G. Myles, C. Collberg // Proceedings of the 2005 ACM symposium on Applied computing. – 2005. – P. 314-318. DOI:10.1145/1066677.1066753.

[5] Karim, M. Malware phylogeny generation using permutations of code / M. Karim, A. Walenstein, A. Lakhotia, L. Parida // Journal in Computer Virology. – 2005. – Vol. 1(1-2). – P. 13-23. DOI: 10.1007/s11416-005-0002-9.

[6] IDA F.L.I.R.T Technology: In-Depth [Electronic resource]. – Access mode: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml.

[7] Bilar, D. Opcodes as predictor for malware // International Journal of Electronic Security and Digital Forensics. – 2007. – Vol. 1(2). – P. 156-168. DOI: 10.1504/IJESDF.2007.016865.

[8] Gheorghescu, M. An automated virus classification system // Virus Bulletin Conference. – 2005. – P. 294-300.

[9] Bruschi, D. Detecting selfmutating malware using control-flow graph matching / D. Bruschi, L. Martignoni, M. Monga // Proceedings of the Third international conference on Detection of Intrusions and Malware & Vulnerability Assessment. – 2006. – P. 129-143. DOI: 10.1007/11790754_8.

[10] Kruegel, C. Polymorphic worm detection using structural information of executables /C. Kruegel, E. Kirda, D. Mutz, W. Robertson, G. Vigna // Recent Advances in Intrusion Detection. – 2005. – P. 207-226. DOI: 10.1007/11663812_11.

[11] Yumaganov, A. A method of searching for similar code sequences in executable binary files using a featureless approach / A. Yumaganov, V. Myasnikov // Computer Optics. – 2017. – Vol. 41(5). – P. 756-764. DOI: 10.18287/2412-6179-2017-41-5-756-764.

[12] Юмаганов, А.С. Поиск похожих последовательностей кода в исполняемых файлах на основе структурного анализа функций / А.С. Юмаганов, В.В. Мясников // Сборник трудов IV международной конференции и молодежной школы «Информационные технологии и нанотехнологии» (ИТНТ) – Самара: Новая техника, 2018. – С. 2429-2436.

[13] Hex-Rays IDA: About [Electronic resource]. – Access mode: http://hex-rays.com/products/ida/ (05.11.2018).

[14] Spath, H. The minisum location problem for the Jaccard metric // Operations-Research-Spektrum. – 1981. – Vol. 3(2). – P. 91-94.

[15] Buckland, M. K. The relationship between recall and precision / M.K. Buckland, F.C. Gey // JASIS. – 1994. – Vol. 45(1). – P. 12-19. DOI: 10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L.

[16] Powers, D.M. W. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation // Journal of Machine Learning Technologies. – 2011. – Vol. 2(1). – P. 37-63.
[17] TIFF Library and Utilities [Electronic resource]. – Access mode: http://www.libtiff.org/ (05.11.2018).

A combined method of similar code sequences search in executable files

A.S. Yumaganov[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. This work is devoted to the development of a method of similar code sequences search in executable files, based on two methods, that implement syntactic code analysis and control flow graph (CFG) analysis respectively. The method of syntactic code analysis is based on the comparison of processor instructions locations in the body of the function. The method of CFG analysis is based on the analysis of CFG's subgraphs of the function. The results of experimental researches of the efficiency of proposed method in comparison with known methods of similar code sequences search are presented.