ЧИСЛЕННЫЕ МЕТОДЫ И АНАЛИЗ ДАННЫХ

МЕТОД ПОИСКА ПОХОЖИХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ КОДА В ИСПОЛНЯЕМЫХ БИНАРНЫХ ФАЙЛАХ С ИСПОЛЬЗОВАНИЕМ БЕСПРИЗНАКОВОГО ПОДХОДА

А.С. Юмаганов[1], В.В. Мясников[1]

[1] Самарский национальный исследовательский университет имени академика С.П. Королева, Самара, Россия

Аннотация

Работа посвящена решению задачи поиска похожих последовательностей кода в исполняемых бинарных файлах. Предлагается метод решения, при котором промежуточное векторное описание функции формируется на основе сравнения пространственного положения каждой из функциональных групп команд процессора данной функции и функций некоторой «базисной» библиотеки. Размерность полученного таким образом вектора понижается, и полученное окончательное описание используется для осуществления поиска. Представлены результаты экспериментальных исследований, демонстрирующие работоспособность данного метода. Исследована эффективность данного метода в сравнении с некоторыми ранее известными методами поиска похожих последовательностей кода, даны рекомендации по выбору параметров разработанного метода.

Ключевые слова: поиск, последовательность кода, беспризнаковое распознавание.

Цитирование: **Юмаганов, А.С.** Метод поиска похожих последовательностей кода в исполняемых бинарных файлах с использованием беспризнакового подхода / А.С. Юмаганов, В.В. Мясников // Компьютерная оптика. – 2017. – Т. 41, № 5. – С. 756-764. – DOI: 10.18287/2412-6179-2017-41-5-756-764.

Введение

Повторное использование кода (code reuse) часто применяется программистами в процессе создания нового программного обеспечения (ПО). Согласно исследованиям [1] около 15 % функций в программах с открытым исходным кодом являются повторно использованными. Такой подход позволяет разработчикам нового ПО экономить время, финансовые и трудовые ресурсы на разработке программных модулей, требуемый функционал которых ранее уже был реализован другими разработчиками. Однако повторное использование кода имеет некоторые недостатки. Во-первых, повторное использование кода может нарушать лицензионное соглашение на использование ПО. Во-вторых, велик риск переноса участка кода, содержащего уязвимость или ошибку. Решение задачи поиска похожих последовательностей кода может быть применено и для выявления нарушений лицензионных соглашений, и для нахождения ошибок и уязвимостей ПО, и для обнаружения вредоносного ПО.

К настоящему моменту известно большое количество методов поиска похожих последовательностей кода в исполняемых файлах. Одним из наиболее известных является алгоритм FLIRT (Fast Library Identification and Recognition Technology) [2], используемый для поиска библиотечных функций в дизассемблере IDA. Он основан на сравнении шаблонов функций, которые представляют собой первые 32 байта функции с пометкой всех изменяемых байтов. Однако данный метод не предполагает идентификацию модифицированных функций. Другой метод, представленный в работе [3], основан на сравнении ограниченных последовательностей команд процессора внутри функций (k-грамм). Данный алгоритм чувствителен к некоторым видам обфускации кода (например, к вставке «мусорных» команд) и оптимизациям компилятора.

Помимо методов поиска, основанных на анализе непосредственно ассемблерного кода, существуют также алгоритмы поиска похожих последовательностей кода на основе сравнения графов потока управления функций [4, 5, 6]. Эти методы чувствительны к структурным изменениям кода и не пригодны для сравнения функций с малым количеством вершин графа потока управления.

Таким образом, несмотря на большое количество научных работ, посвященных проблеме поиска похожих последовательностей кода, не существует универсального метода решения данной проблемы, лишенного всех недостатков.

В данной статье предлагается и исследуется метод поиска функций исполняемого бинарного файла, схожих с известными функциями из некоторого «архива» программ. Данный метод можно использовать, например, для выявления нарушений лицензионного соглашения, если «архив» содержит функции определенной библиотеки, нарушение лицензионного соглашения которой нужно установить. Учитывая чрезвычайную сложность получения всеобъемлющего описания кода (графа) исполнения, предлагаемый метод поиска основан на *принципах беспризнакового распознавания* – интересующая нас функция представляется через ее отношения (схожести/близости) с функциями из некоторой базисной библиотеки. Получаемое таким образом избыточное описание для реализации последующего поиска сокращается с помощью метода главных компонент (англ.: Principle Component Analysis – PCA). Идеи предлагаемого метода и некоторые элементы его

реализации были представлены в работах [7] и [8] авторов. Настоящая работа существенно обобщает и дополняет их результаты.

Работа построена следующим образом. В первом параграфе даны основные определения и краткое описание предложенного метода. Во втором параграфе рассматривается процесс получения первичного описания функции. Третий параграф посвящен способам построения промежуточного описания функции через набор вспомогательных функций (*базисную библиотеку*). В четвертом параграфе кратко описан метод снижения размерности, применяемый для получения окончательного представления функции. В пятом параграфе представлен алгоритм поиска похожих функций на основе полученного описания. В шестом параграфе приводится способ оценки эффективности метода и результаты проведенных экспериментов. В заключении приводятся выводы, представлен список использованной литературы.

1. Основные понятия и принцип работы

Введем следующие базовые понятия:

– текущая библиотека – набор функций исследуемого исполняемого файла;

– архивные данные – набор известных функций и их описаний через библиотеку базисных функций;

– базисная библиотека – вспомогательный набор функций, применяемый для сравнения функций архивных данных и текущей библиотеки.

Решаемая задача формулируется следующим образом: для заданной (или каждой) функции текущей библиотеки найти наиболее похожую функцию из архивных данных. Конкретизация понятия (меры) сходства в разных задачах и разных вариантах решения может быть, вообще говоря, различным. В рамках предлагаемого метода рассматриваются и исследуются несколько вариантов меры сходства, основанных на расположении команд определенных функциональных групп в теле функции. Подробное описание каждого из них представлено в третьем параграфе.

Решение сформулированной выше задачи предлагается осуществлять в несколько этапов. На первом этапе производится представление функций архивных данных через отношения (похожести) с функциями некоторой наперед выбранной библиотекой базисных функций. Получаемое представление сокращается с использованием метода PCA и заносится в архивную базу данных (БД). На втором этапе аналогичным образом формируется представление функций текущей библиотеки, которое затем записывается в соответствующую БД. Непосредственно поиск на третьем этапе реализуется средствами БД, выполняя упорядочивание описаний функций архивных данных по критерию евклидовой близости.

Предлагаемый метод, состоящий из указанных трех этапов (параграфы 2–5), имеет ряд настроечных параметров, которые можно использовать для повышения эффективности предлагаемого решения.

2. Построение первичного описания функций

С помощью дизассемблера можно провести анализ кода исследуемого исполняемого файла и получить код ассемблера данного файла с разбиением на функции, каждая из которых состоит из набора команд процессора и операндов.

Все команды процессора разбиваются на $K = 47$ функциональных групп [9]. Каждая группа включает в себя множество команд, предназначенных для выполнения однотипных операций. Примерами таких групп являются: группа команд пересылки данных (Data Transfer Instructions), группа команд логических операций (Logical Instructions). Такое разбиение предъявляет требование к используемой базисной библиотеке: каждая из K функциональных групп команд должна быть представлена как минимум в одной из функций библиотеки базисных функций.

Рассмотрим некоторую функцию. Каждой из K функциональных групп для этой функции ставится в соответствие список смещений относительно начала функции, на которых расположены команды данной группы (если они существуют). Полученное таким образом первичное описание функции заносится в БД и впоследствии используется для построения промежуточного описания через библиотеку базисных функций.

3. Построение промежуточного описания функций

Для построения промежуточного описания рассматриваемой функции через библиотеку базисных функций необходимо получить значения некоторой меры схожести с каждой из базисных функций. В качестве объекта сравнения для получения этой меры будем использовать один из следующих:

1) пространственное распределение команд для каждой функциональной группы в дифференциальной форме (гистограмма);

2) пространственное распределение команд для каждой функциональной группы в интегральной форме;

3) ядерную оценку пространственного распределения команд для каждой функциональной группы;

4) длину наибольшей общей подпоследовательности списка номеров групп команд.

Рассмотрим подробно процесс получения каждого из представленных выше объектов сравнения и процесс формирования первичного описания исследуемой функции.

Пространственное распределение команд для каждой функциональной группы в дифференциальной (гистограмма) и интегральной форме

Для каждой группы команд исследуемой функции строится пространственное распределение команд, входящих в эту группу, по длине функции следующим образом.

Пусть $n_0^k,\ldots,n_{N_k-1}^k$ – абсолютные смещения (позиции) относительно начала функции команд группы k, N_k – общее количество команд этой группы в данной функции, N – длина функции. Определим простран-

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

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

здесь $I(.)$ – индикатор события, принимающий значения «0» или «1» в зависимости от истинности соответствующего аргумента.

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

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

Выбрав одну из форм представления пространственного распределения команд в теле функции, получаем *K* векторов приведенного ниже вида, каждый из которых характеризует пространственное положение команд *k*-й группы в теле функции:

$$\overline{a}_k = \left(f_1^k, f_2^k, \ldots, f_I^k\right)^T, k = \overline{0, K-1}. \tag{3}$$

Данный набор векторов совместно формирует матрицу описания:

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

Аналогичную матрицу, именуемую далее *C*, можно построить для каждой функции, в том числе и для функций базисной библиотеки.

Мерой схожести двух функций, задаваемых матрицами *A* и *B*, назовем следующую величину:

$$\mu\left(A, C\right) = \sum_{k=0}^{K-1} \alpha_k \mu_{\cos}(\overline{a}_k, \overline{c}_k), \tag{5}$$

где

$$\mu_{\cos}(\overline{a}, \overline{c}) = \frac{\sum_{i=1}^{I} a_i c_i}{\sqrt{\sum_{i=1}^{I} a_i^2}\sqrt{\sum_{i=1}^{I} c_i^2}},$$

а величины $\alpha_k \geq 0$, удовлетворяющие условию $\sum_{k=0}^{K-1} \alpha_k = 1$, – суть параметры метода. При полном совпадении функций мера схожести принимает значение «1», при полном несоответствии – «0».

Пусть далее *J* – число функций в базисной библиотеке, каждая из которых имеет описание в виде матрицы C_j. Тогда, сравнивая исследуемую функцию текущей библиотеки с каждой из функций базисной библиотеки, получим следующий вектор промежуточного описания этой функции:

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

Ядерная оценка пространственного распределения команд для каждой функциональной группы

Для устранения влияния на результат поиска малых смещений команд в коде при использовании формулы (1) воспользуемся ядерной оценкой плотности распределения (kernel density estimation) [10]. Тогда оценка распределения команд в коде имеет вид:

$$f_i^k = \sum_{j=1}^{N} \tilde{f}_j^k \frac{1}{h} K\left(\frac{i-j}{h}\right), \quad i = \overline{0, I},$$

где $K(r)$ – функция ядра, *h* – ширина окна. В качестве функции ядра может выступать, например, ядро Гаусса:

$$K(r) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} r^2\right).$$

Как и в предыдущем параграфе, воспользуемся формулами (3), (4), (5) для получения представления рассматриваемой функции через библиотеку базисных функций в виде вектора (6).

Длина наибольшей общей подпоследовательности (НОП)

Используя полученное ранее первичное описание функции, получим упорядоченную по смещениям относительно начала функции последовательность номеров групп соответствующих команд для рассматриваемой функции. Аналогичным образом получим последовательность номеров групп для функций базисной библиотеки.

Рассмотрим последовательность чисел $a_1, a_2, \ldots, a_n$ (далее, для краткости, используем обозначение $\{a_n\}$). Если из данной последовательности чисел удалить часть элементов, то получим новую последовательность, которую называют подпоследовательностью исходной последовательности. Рассмотрим еще одну последовательность чисел $b_1, b_2, \ldots, b_n$ (далее, для краткости, используем обозначение $\{b_n\}$). Пусть требуется найти длину самой длинной подпоследовательности последовательности $\{a_n\}$, которая одновременно является и подпоследовательностью последовательности $\{b_n\}$. Такую последовательность называют *наибольшей общей подпоследовательностью* (НОП) двух последовательностей [11].

Пусть последовательность $\{a_n\}$ номеров групп первой функции имеет длину N_1, а последовательность $\{b_n\}$ номеров групп второй функции – N_2. Мерой схожести двух функций, задаваемых последовательностями *a*,*b*, назовем следующую величину:

$$\mu\left(\{a_n\}, \{b_n\}\right) = \frac{2 * ДНОП(\{a_n\}, \{b_n\})}{N_1 + N_2},$$

где ДНОП – длина НОП. При полном совпадении функций мера схожести принимает значение «1», при полном несоответствии – «0».

Для решения задачи нахождения ДНОП воспользуемся алгоритмом Хиршберга [11]. Работа алгоритма сводится к поэтапному заполнению матрицы *L*, строки которой представляют собой элементы после-

довательности $\{a_n\}$, а столбцы – элементы последовательности $\{b_n\}$.

Матрица L заполняется следующим образом:

$$L(i,j)=\begin{cases}0, i=0\vee j=0,\\ L(i-1,j-1)+1, a(i)=b(j),\\ \max(L(i-1,j),L(i,j-1)), a(i)\neq b(j).\end{cases}$$

Таким образом, в ячейке *(i,j)* матрицы L хранится длина НОП для последовательностей $\{a_n\}_{n=1}^{i}$ и $\{b_n\}_{n=1}^{j}$. В результате работы алгоритма получим матрицу L, элемент $L(N_1, N_2)$ которой содержит искомую длину НОП двух последовательностей.

Пусть J – число функций в библиотеке базисных функций. Сравнивая исследуемую функцию с каждой из функций библиотеки базисных функций, получим следующий вектор описания данной функции:

$$\overline{x}_{\{a_n\}}=\left(\mu\left(\{a_n\},\{b^0{}_n\}\right),\ldots,\mu\left(\{a_n\},\{b^{J-1}{}_n\}\right)\right)^T, \quad (7)$$

который назовем *вектором промежуточного описания* рассматриваемой функции.

4. Построение окончательного описания функций

В предыдущем параграфе было получено промежуточное описание рассматриваемой функции в виде вектора (6) или (7). Так как количество функций в библиотеке базисных функций $J>>1$, воспользуемся известным методом снижения размерности – методом главных компонент [12].

Этот метод сводится к вычислению собственных векторов и собственных значений ковариационной матрицы соответствующих векторов:

$$B=\mathrm{E}\{(\overline{x}-\mathrm{E}(\overline{x}))(\overline{x}-\mathrm{E}(\overline{x}))^T\}.$$

Выполним сортировку собственных значений ковариационной матрицы по убыванию и расположим в таком же порядке соответствующие им собственные векторы. Возьмем $I<J$ максимальных собственных значений, тогда получим матрицу перехода к новой размерности Y, составленную из I собственных векторов. Для получения вектора признаков размерности $I<J$ воспользуемся следующим выражением (индекс, содержащий ссылку на первоначальное описание, далее опустим):

$$\overline{z}=Y\overline{x}. \quad (8)$$

Вектор z выступает в качестве *окончательного представления исследуемой функции* через набор функций базисной библиотеки и не содержит элементов первичного описания, что позволяет предлагаемый подход ассоциировать с *беспризнаковыми методами распознавания* [13].

Изложенный выше способ получения окончательного описания функции применятся как для функций архивных данных, так и для функций текущей библиотеки.

5. Поиск похожих функций

Получив окончательное представление функций текущей библиотеки и функций архивных данных в виде вектора (8), переходим к заключительному этапу предлагаемого метода, в котором осуществляется непосредственно поиск похожих функций.

Пусть $\overline{z}$ – вектор окончательного описания функции текущей библиотеки через библиотеку базисных функций, $\overline{z}^*$ – вектор окончательного описания архивной функции через библиотеку базисных функций. Для сравнения векторов признаков воспользуемся евклидовым расстоянием:

$$d\left(\overline{z},\overline{z}^*\right)=\sqrt{\sum_{i=0}^{I-1}(z_i-z_i^*)^2}, \quad (9)$$

где z_i – значение i-й компоненты вектора признаков первой функции, z_i^* – значение i-й компоненты вектора признаков второй функции. При $d=0$ функции считаются одинаковыми.

Для уменьшения времени, затрачиваемого на поиск похожих функций, положим, что при модификации функции ее размер изменяется не более чем на 30 %. Тогда для поиска функций, похожих на данную, среди архивных функций воспользуемся следующим алгоритмом:

- Определяем длину рассматриваемой функции N.
- Получаем список функций архивных данных, размер которых отличается от исследуемой функции не более чем на 30 %.
- Для каждой функции из списка, полученного на втором шаге, найдем евклидово расстояние (9) до исследуемой функции.
- Сортируем полученный на предыдущем шаге результат по возрастанию величины d.

В результате для рассматриваемой функции получаем упорядоченный по уменьшению схожести список функций архивных данных.

6. Результаты экспериментов

Программная реализация представленного метода поиска была выполнена на языке Python. Для проведения экспериментальных исследований использовался стандартный ПК (Intel Core i5-3450 3.1 ГГц, 16 Гб ОЗУ).

Для проверки эффективности предложенного метода поиска схожих функций возьмем в качестве архивных данных функции одной динамической библиотеки, а в качестве текущей – функции такой же библиотеки, только другой версии. Установим минимальный размер функции, равный 16 байтам, так как рассматриваемые библиотеки содержат большое количество «маленьких» функций (1–3 инструкции), содержащих одинаковые команды. Для определения априори похожих (идентичных) функций считалось, что при модификации кода динамической библиотеки (при переходе от одной версии к другой) имена функций не менялись и среди функций архивных данных нет функций с одинаковым именем.

Для проверяемой функции текущей библиотеки получим список похожих функций архивных данных $\{F_l\}_{l=1,\ldots,L}$, отсортированный по уменьшению схожести.

При указанных допущениях оценить качество распознавания можно, сопоставив этому списку функций бинарную последовательность $\beta = (\beta_1, \beta_2, \ldots, \beta_L)$, такую, что при наличии на l-й позиции функции с тем же именем $\beta_l = 1$, иначе – $\beta_l = 0$. Введем следующие меры, используемые обычно как критерии качества информационного поиска [14], [15].

Точность для k-й позиции списка P_k:

$$P_k = \frac{\sum_{l=1}^{k} \beta_l}{k}$$

характеризует отношение количества функций, имеющих то же имя, что и исследуемая функция на первых k позициях упорядоченного списка к общему количеству функций в списке.

Полнота для k-й позиции списка R_k:

$$R_k = \frac{\sum_{l=1}^{k} \beta_l}{K}$$

характеризует отношение количества функций имеющих то же имя, что и исследуемая функция на первых k позициях упорядоченного списка к общему количеству функций с тем же именем среди архивных функций.

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 текущей библиотеки* вычисляется по формуле:

$$P = \frac{1}{S} \sum_{s=0}^{S-1} AveP_s , \qquad (10)$$

где S – количество функций в текущей библиотеке.

Пусть архивные данные представлены динамической библиотекой libtiff 4.0.3 [16], а текущая библиотека представлена одной из следующих версий библиотеки libtiff: libtiff 3.9.7, libtiff 4.0.4, libtiff 4.0.5, libtiff 4.0.6. В каждой из данных библиотек содержится около 1200 функций, размер которых превышает 16 байт.

Библиотека базисных функций содержит $J = 50$ функций, выбранных вручную из различных исполняемых файлов. Для каждой из $K = 47$ функциональных групп команд существует как минимум одна функция базисной библиотеки, содержащая команды этой группы.

Вектор окончательного описания функции через библиотеку базисных функций имеет размер $I = 5$.

Основным параметром представленного метода поиска похожих функций является выбор сравниваемой меры (объекта сравнения). Как отмечалось ранее, в качестве такой меры могут выступать:

- пространственное распределение команд в теле исследуемой функции (в дифференциальной или в интегральной форме);
- ядерная оценка пространственного распределения команд в теле исследуемой функции;
- длина наибольшей общей подпоследовательности групп команд в теле функции.

В свою очередь, при нахождении ядерной оценки пространственного распределения команд в теле функции требуется определить вид функции ядра и значение параметра h.

Рассмотрим влияние упомянутых выше параметров ядерной оценки пространственного распределения команд в теле функции на эффективность представленного в работе метода поиска похожих функций при соответствующе заданном объекте сравнения.

На практике выбор функции ядра является менее важной задачей, нежели выбор значения параметра h. Это обосновано тем, что, используя разные функции ядра, можно получить достаточно близкие оценки плотности распределения, подстраивая значение параметра h с помощью канонических значений h (canonical bandwidths) [17]. По этой причине рассмотрим подробнее выбор значения параметра h при фиксированной функции ядра (ядро Гаусса).

На рис. 1 показан график зависимости средней точности поиска (10) от значения параметра h при Гауссовой функции ядра. Как видно из графика, максимальное значение средней точности поиска достигается при $h = 6{,}5$.

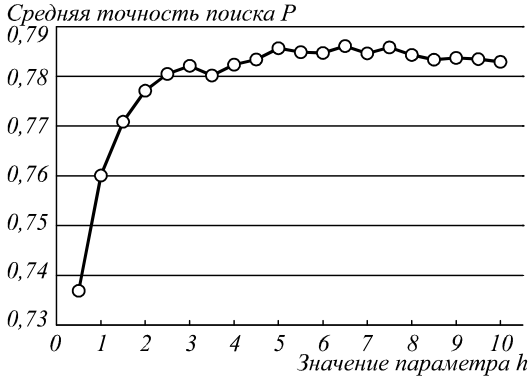

Рис. 1. Зависимость средней точности поиска P от значения параметра h

На рис. 1 представлены результаты для библиотеки libtiff версии 3.9.7 (в качестве текущей библиотеки). Однако следует отметить, что оптимальное значение параметра h (то, которое соответствует максимальному значению средней точности поиска) зависит от размера выборки, а следовательно, и от количества команд какой-либо группы внутри рассматриваемой функции и функций базисной библиотеки. Это значит, что при использовании в качестве библиотеки базисных функций другого набора функций (отличного от применяемого в этой работе) и/или другого набора функций текущей библиотеки оптимальное значение параметра h может быть другим.

Таким образом, при оценивании точности поиска, где в качестве меры сравнения используется ядерная оценка пространственного распределения команд, значение параметра h для каждой из рассматриваемых текущих библиотек было подобрано опытным путем. В табл. 1 представлены используемые в дан-

ной работе значения h в зависимости от выбора текущей библиотеки при Гауссовой функции ядра.

Сравним значения средней точности поиска для четырех версий библиотеки libtiff при использовании рассмотренных выше объектов сравнения. В табл. 2 представлены полученные результаты.

Табл. 1. Выбор параметра h

Текущая библиотека	Значение параметра h
libtiff 3.9.7	6,5
libtiff 4.0.4	3,5
libtiff 4.0.5	3,5
libtiff 4.0.6	3,0

Табл. 2. Сравнение точности поиска для различных объектов сравнения

Текущая библиотека	Средняя точность поиска P для заданного объекта сравнения			
	Пространственное распределение команд в теле функции в дифференциальной форме	Пространственное распределение команд в теле функции в интегральной форме	Ядерная оценка пространственного распределения команд в теле функции	Длина НОП групп команд
libtiff 3.9.7	0,7087	0,7854	0,7865	0,7276
libtiff 4.0.4	0,8406	0,8515	0,8550	0,8126
libtiff 4.0.5	0,8377	0,8499	0,8514	0,8117
libtiff 4.0.6	0,8369	0,8428	0,8391	0,8052

Наименьшая средняя точность поиска в большинстве случаев получается при использовании в качестве объекта сравнения длины наибольшей общей подпоследовательности (ДНОП) групп команд. Это связано с тем, что в случае сравнения на основе пространственного распределения команд в теле функции учитывается не только последовательность команд, но и размеры инструкций и их операндов, так как значение длины функции в формуле (1) зависит от этих значений. В случае ДНОП учитывается только порядок инструкций в теле рассматриваемой функции, что снижает точность поиска похожих функций. Лучший результат в трех из четырех вариантов текущей библиотеки был получен при использовании ядерной оценки пространственного распределения команд в качестве объекта сравнения.

Результаты экспериментов показали, что лучшим объектом сравнения (параметр предложенного метода) по критерию максимальной средней точности поиска является ядерная оценка пространственного распределения команд в теле функции, по критерию времени – пространственное распределение команд в теле функции в дифференциальной форме. Однако выбор параметра h для ядерной оценки распределения зависит от функций текущей библиотеки, что затрудняет использование данного объекта сравнения. Этого недостатка лишено сравнение на основе пространственного распределения команд в теле функции (в дифференциальной или интегральной форме), что делает использование этих объектов сравнения наиболее удобными.

Рассмотрим работу данного метода при сравнении двух совершенно разных исполняемых файлов. Пусть архивные данные представлены функциями библиотеки curl [18] версии 7.49.0. На рис. 2 представлена гистограмма значений евклидовой метрики для заданных архивных данных при использовании в качестве текущей библиотеки функций либо другой версии той же библиотеки (curl 7.53.1), либо совершенно другого исполняемого файла – «calc.exe» (стандартное приложение Windows 7). Значение евклидовой метрики для функции текущей библиотеки соответствует первому элементу списка похожих функций архивных данных для заданной функции, алгоритм получения которого описан в пятом параграфе настоящей работы. Как видно из рисунка, евклидово расстояние между векторами описания функций архивной библиотеки и подавляющим большинством функций исполняемого файла «calc.exe» существенно выше, чем между функциями разных версий одной библиотеки curl. Этот факт подтверждает работоспособность представленного метода поиска похожих функций.

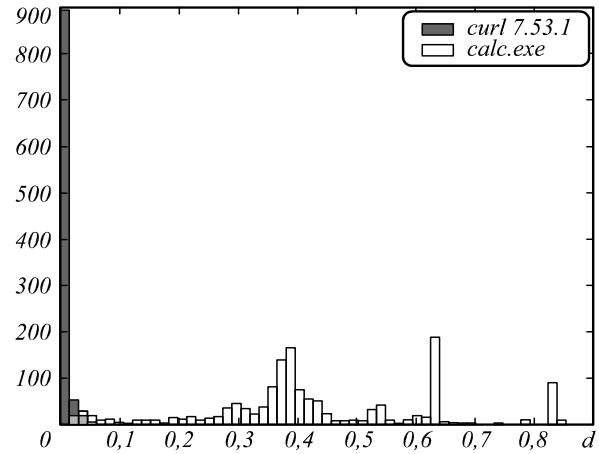

Рис. 2. Гистограмма значений евклидового расстояния (9) для двух различных текущих библиотек (архивные данные – curl 7.49.0)

Рассмотрим *среднее время*, затрачиваемое на построение архивной и текущей баз данных, для каждого из рассмотренных объектов сравнения. В табл. 3 представлены полученные результаты. Минимальное время построения баз данных получается при использовании в качестве объекта сравнения пространственного распределения команд в теле функции в дифференциальной форме.

При использовании в качестве объекта сравнения длины НОП групп команд время, затрачиваемое на построение БД, увеличивается почти в 5 раз по сравнению с лучшим результатом. Другие два объекта сравнения (пространственное распределение команд в интегральной форме и ядерная оценка пространственного распределения команд) показали результат, уступающий лидеру примерно в 3 раза.

Табл. 3. Сравнение времени построения БД

Объект сравнения	Среднее время построения БД, с
Пространственное распределение команд в теле функции в дифференциальной форме	342
Пространственное распределение команд в теле функции в интегральной форме	1023
Ядерная оценка пространственного распределения команд в теле функции	1135
Длина НОП групп команд	1502

Сравним *эффективность представленного в данной работе метода поиска похожих функций* (объект сравнения – пространственное распределение команд в теле функции в интегральной форме) и других известных методов поиска похожих функций: алгоритма распознавания библиотечных функций IDA FLIRT [2] и метода, основанного на сравнении k-грамм [3]. Для метода, основанного на сравнении k-грамм, параметр k был выбран равным 5 согласно рекомендациям авторов [3]. Полученные результаты представлены в табл. 4.

Анализ данных табл. 4 показывает, что представленный в данной работе метод поиска похожих функций не уступает названным выше алгоритмам по показателю средней точности поиска P, а в большинстве случаев превосходит их.

Табл. 4. Сравнение методов поиска похожих функций

Текущая библиотека	Средняя точность поиска P похожих функций, используя разработанный метод	Средняя точность поиска P похожих функций, используя алгоритм IDA FLIRT	Средняя точность поиска P похожих функций, используя метод, основанный на сравнении k-грамм функции (k=5)
libtiff 3.9.7	0,7854	0,5035	**0,8068**
libtiff 4.0.4	**0,8515**	0,8006	0,8249
libtiff 4.0.5	**0,8499**	0,7912	0,8217
libtiff 4.0.6	**0,8428**	0,7656	0,8189

Заключение

В работе представлен метод поиска похожих последовательностей кода в исполняемых бинарных файлах. В данном методе описание функций формируется на основе сравнения пространственного положения команд процессора рассматриваемой функции и функций, составляющих некоторую базисную библиотеку. Приводятся результаты экспериментальных исследований предложенного метода, демонстрирующие его работоспособность и эффективность. Продемонстрировано превосходство оригинального метода над некоторыми известными алгоритмами поиска похожих последовательностей кода в исполняемых бинарных файлах. Также даны рекомендации по выбору некоторых параметров разработанного метода.

Литература

1. **Zaimi, A.** An empirical study on the reuse of third-party libraries in open-source software development / A. Zaimi, A. Ampatzoglou, N. Triantafyllidou, A. Chatzigeorgiou, A. Mavridis, T. Chaikalis, I. Deligiannis, P. Sfetsos, I. Stamelos // Proceedings of the 7th Balkan Conference on Informatics Conference. – 2015. – 4. – DOI: 10.1145/2801081.2801087.
2. IDA F.L.I.R.T Technology: In-Depth [Electronical Resource]. URL: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml (request date 6.03.2017).
3. **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.
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.** Polymorphic worm detection using structural information of executables / C. Kruegel, E. Kirda // Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection. – 2005. – P. 207-226. – DOI: 10.1007/11663812_11.
6. **Khoo, W.M.** Rendezvous: A search engine for binary code / W.M. Khoo, A. Mycroft, R. Anderson // Proceedings of the 10th Working Conference on Mining Software Repositories. – 2013. – P. 329-338. – DOI: 10.1109/MSR.2013.6624046.
7. **Yumaganov, A.S.** Similarity search over program code sequences using featureless pattern recognition techniques / A.S. Yumaganov, V.V. Myasnikov // CEUR Workshop Proceedings. – 2016. – Vol. 1638. – P. 437-443. – DOI: 10.18287/1613-0073-2016-1638-437-443.
8. **Юмаганов, А.С.** Сравнение способов первичного описания кода программы в задаче поиска похожих последовательностей кода / А.С. Юмаганов, В.В. Мясников // Сборник трудов III Международной конференции и молодежной школы «Информационные технологии и нанотехнологии» (ИТНТ-2017) – Самара: Новая техника, 2017. – С. 940-945.
9. x86 Assembly language reference manual [Electronical Resource]. – 2010. – URL: https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf (request date 06.03.2017).
10. **Фукунага, К.** Введение в статистическую теорию распознавания образов: пер. с англ. / К. Фукунага. – М.: Наука, 1979. – 368 с.
11. **Hirschberg, D.S.** A linear space algorithm for computing maximal common subsequences / D.S. Hirschberg // Communications of the ACM. – 1975. – Vol. 18, Issue 6. – P. 341-343. – DOI: 10.1145/360825.360861.
12. **Pearson, K.** On lines and planes of closest fit to systems of points in space / K. Pearson // Philosophical Magazine. – 1901 . – Vol. 2. – P. 559-572.
13. **Duin, R.P.W.** Featureless pattern classification / R.P.W. Duin, D. de Ridder, D.M.J. Tax // Kybernetica. – 1998. – Vol. 34, No. 4. – P. 399-404.

14. **Buckland, M.K.** The relationship between recall and precision / M.K. Buckland, F.C. Gey // Journal of the American Society for Information Science. – 1994. – Vol. 45, Issue 1. – P. 12-19. – DOI: 10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L.
15. **Powers, D.M.W.** Evaluation: From precision, recall and f-measure to ROC, informedness, markedness & correlation / D.M.W. Powers // Journal of Machine Learning Technologies. – 2011. – Vol. 2, Issue 1. – P. 37-63.
16. LibTIFF – TIFF library and utilities [Electronical Resource]. – URL: http://www.libtiff.org/ (request date 6.03.2017).
17. **Marron, J.S.** Canonical kernels for density estimation / J.S. Marron, D. Nolan // Statistics & Probability Letters. – 1989. – Vol. 7, Issue 3. – P. 195-199. – DOI: 10.1016/0167-7152(88)90050-8.
18. Curl – Command line tool and library for transferring data with URLs [Electronical Resource]. – URL: https://curl.haxx.se/ (request date 19.06.2017).

Сведения об авторах

Юмаганов Александр Сергеевич, 1993 года рождения. В 2016 году окончил Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет) по специальности «Информационная безопасность автоматизированных систем». В настоящее время является аспирантом Самарского университета. Область научных интересов: программирование, распознавание образов. Имеет 2 публикации. E-mail: *yumagan@gmail.com* .

Мясников Владислав Валерьевич, 1971 года рождения. В 1994 году окончил Самарский государственный аэрокосмический университет (СГАУ). В 1995 году поступил в аспирантуру СГАУ, в 1998 году защитил диссертацию на соискание степени кандидата технических наук, а в 2008 – диссертацию на соискание степени доктора физико-математических наук. В настоящее время работает профессором кафедры геоинформатики и информационной безопасности Самарского университета и ведущим научным сотрудником в Институте систем обработки изображений РАН – филиале федерального государственного учреждения «Федеральный научно-исследовательский центр «Кристаллография и фотоника» РАН». Круг научных интересов включает цифровую обработку сигналов и изображений, компьютерное зрение, распознавание образов, искусственный интеллект и геоинформатику. Имеет более 160 публикаций, в том числе около 80 статей и две монографии (в соавторстве). Член Российской ассоциации распознавания образов и анализа изображений. Страница в интернете: http://www.ipsi.smr.ru/staff/MyasVV.htm. E-mail: *vmyas@smr.ru* .

ГРНТИ: 28.23.15.
Поступила в редакцию 22 мая 2017 г. Окончательный вариант – 23 июня 2017 г.

A METHOD OF SEARCHING FOR SIMILAR CODE SEQUENCES IN EXECUTABLE BINARY FILES USING A FEATURELESS APPROACH

A.S. Yumaganov[1], V.V. Myasnikov[1]
[1] Samara National Research University, Samara, Russia

Abstract

The work is devoted to solving a problem of searching for similar code sequences in executable binary files. The proposed method involves partitioning the processor instructions into functional groups, forming a given function's primary description by commands position in its body, followed by generating the function's intermediate description through its comparison with the functions from a "base" library. With the dimensionality of the resulting vector reduced in this way, the resulting final description is then used to perform the search. Results of the experimental study demonstrate the operability of the proposed method. The efficiency of the proposed method is compared against existing methods of searching for similar code sequences. We also provide recommendations on the choice of parameters of the developed method.

Keywords: searching, code sequence, featureless recognition.

Citation: 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.

References

[1] Zaimi A, Ampatzoglou A, Triantafyllidou N, Chatzigeorgiou A, Mavridis A, Chaikalis T, Deligiannis I, Sfetsos P, Stamelos I. An empirical study on the reuse of third-party libraries in open-source software development. Proceedings of the 7th Balkan Conference on Informatics Conference 2015: 4. DOI:10.1145/2801081.2801087.
[2] IDA F.L.I.R.T Technology: In-Depth. Source: ⟨https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml⟩.
[3] Myles G, Collberg C. K-gram based software birthmarks. Proceedings of the 2005 ACM Symposium on Applied Computing 2005; 314-318. DOI: 10.1145/1066677.1066753.

[4] Flake H. Structural comparison of executable objects. Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment 2004; 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] Khoo WM, Mycroft A, Anderson R. Rendezvous: A search engine for binary code. MSR '13 2013; 329-338. DOI: 10.1109/MSR.2013.6624046.
[7] 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.
[8] Yumaganov AS, Myasnikov VV. Comparison of the ways of the program's code initial description in the problem of similar code sequences search [In Russian]. Proceedings of the III International Conference and Youth School ITNT-2017. Samara: "Novaya Tehnika" Publisher; 2017: 940-945.
[9] x86 Assembly language reference manual. Source: ⟨https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf⟩.
[10] Fukunaga K. Introduction to statistical pattern recognition. 2nd ed. San Diego, London, San Francisco: Academic Press; 1990. ISBN: 978-0-08-047865-4.
[11] Hirschberg DS. A linear space algorithm for computing maximal common subsequences. Communications of the ACM 1975; 18(6): 341-343. DOI: 10.1145/360825.360861.
[12] Pearson K. On lines and planes of closest fit to systems of points in space. Philosophical Magazine 1901; 2: 559-572.
[13] Duin RPW, de Ridder D, Tax DMJ. Featureless pattern classification. Kybernetica 1998; 34(4): 399-404.
[14] Buckland MK, Gey FC. The relationship between recall and precision. J Am Soc Inf Sci 1994; 45(1): 12-19. DOI: 10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L.
[15] Powers DMW. Evaluation: From precision, recall and f-measure to ROC, informedness, markedness & correlation. Journal of Machine Learning Technologies 2011; 2(1): 37-63.
[16] LibTIFF – TIFF library and utilities. Source: ⟨http://www.libtiff.org/⟩.
[17] Marron JS, Nolan D. Canonical kernels for density estimation. Statistics & Probability Letters 1989; 7(3): 195-199. DOI: 10.1016/0167-7152(88)90050-8.
[18] Curl – Command line tool and library for transferring data with URLs. Source: ⟨https://curl.haxx.se/⟩.

Authors' information

Alexander Sergeevich Yumaganov (b. 1993) graduated from Samara National Research University (2016), programme – Information Security of Automated Systems. Nowadays he is postgraduate at Samara National Research University. His interests include computer programming, pattern recognition. He has two publications. E-mail: *yumagan@gmail.com* .

Vladislav Valerievich Myasnikov (b. 1971), graduated (1994) from the S.P. Korolyov Samara State Aerospace University (SSAU). He received his PhD in Technical Sciences (2002) and DrSc degree in Physics & Maths (2008). At present he is a professor at the Geoinformatics and information Security department of Samara National Research University and at the Image Processing Systems Institute of the Russian Academy of Sciences. The area of interests includes digital signals and image processing, geoinformatics, neural networks, computer vision, pattern recognition and artificial intelligence. He's list of publications contains about 160 scientific papers, including 80 articles and 2 monographs. He is a member of Russian Association of Pattern Recognition and Image Analysis. Web-page: http://www.ssau.ru/staff/62061001. E-mail: *vmyas@smr.ru* .

Received May 22, 2017. The final version – June 23, 2017.