Поиск похожих последовательностей кода в исполняемых файлах с помощью сиамской нейронной сети

А.С. Юмаганов[1]

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

Аннотация. Данная работа посвящена решению задачи поиска похожих последовательностей кода (функций) в исполняемых файлах. Описание функций, получаемых с помощью предложенного метода решения данной задачи, строятся на основе взаимного пространственного положения команд процессора и соответствующих им операндов в теле функции. Для формирования промежуточного описания функций исполняемого файла используется известная модель векторного представления слов word2vec. Окончательное описание функций формируется с помощью сиамской нейронной сети долгой краткосрочной памяти, которое затем непосредственно используется для поиска схожих функций. Представлены результаты экспериментальных исследований разработанного метода в сравнении с некоторыми известными ранее методами.

1. Введение

В настоящее время при создании нового программного обеспечения (ПО) разработчики часто используют код, разработанный ранее для решения других задач. Повторное использование кода позволяет уменьшить время на разработку нового ПО, но так же может стать причиной возникновения ошибок и уязвимостей в созданном продукте. Согласно исследованиям представленным в [1], 69 уязвимых фрагментов C++ кода, найденных с сайта StackOverflow.com, были использованы в 2859 проектах на сервисе GitHub. Кроме того, использование чужого кода может быть незаконным.

Исходный код большинства выпускаемых программных продуктов не является открытым, что затрудняет его анализ. В этом случае анализ исполняемых файлов является единственным способом проведения анализа кода программного продукта.

Таким образом, решение задачи поиска похожих последовательностей кода в исполняемых файлах позволяет решить такие задачи, как нахождение известных уязвимостей и поиск плагиата, не используя исходный текст анализируемого ПО.

Существует множество методов поиска похожих последовательностей кода в исполняемых файлах. В [2] представлен метод поиска похожих последовательностей кода, в основе которого лежит сравнение ограниченных последовательностей команд процессора (k-gramm). Для поиска библиотечных функций в дизассемблере IDA используется алгоритм FLIRT (Fast Library Identification and Recognition Technology) [3], основанный на сравнении шаблонов функций. Названные выше методы чувствительны к синтаксическим изменениям кода программы (например, замена команды на эквивалентную команду или группу команд). Существуют так же методы поиска, основанные на анализе графа потока управления функций.

Авторы [4] представили метод поиска, в котором для сравнения двух функций формируется их векторное представление, учитывающие структуру графа потока управления, и с помощью нейронной сети оценивается степень их схожести. Представленный в работе [5] метод поиска похожих последовательностей кода основан на анализе подграфов графа потока управления функции. Данная группа методов так же не лишена недостатков: чувствительность к структурным изменениям графа потока управления функций, неприменимость к функциям с малым числом базовых блоков в графе потока управления.

В данной работе представлен метод поиска похожих последовательностей кода, в котором первичное описание функций формируется на основе пространственного положения команд процессора и соответствующих им операндов. Для формирования окончательного описания функции используется сиамская нейронная сеть.

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;

С учетом представленных выше определений решаемая задача формулируется следующим образом: для заданной (или каждой) функции текущей библиотеки найти наиболее похожую функцию из архивных данных.

Предлагаемый в данной работе метод поиска похожих последовательностей кода включает в себя несколько этапов. На первом этапе осуществляется обучение сиамской нейронной сети. На втором этапе с помощью обученной нейронной сети формируется описание функций архивных данных. На третьем этапе аналогичным образом формируется описание функций текущей библиотеки. На заключительном этапе осуществляется непосредственно поиск похожих функций.

3. Формирование первичного описания функции

Выполнив анализ исполняемого файла с помощью дизассемблера можно получить код данного файла с разбиением на функции. Каждая функция состоит из последовательности команд процессора и соответствующих им операндов.

Все команды процессора разбиваются на K функциональных групп по типу выполняемых операций (например, группа логических операций, группа арифметических операций). Все операнды так же разбиваются на N групп согласно представленным в дизассемблере IDA [6] типам операндов (например, базовый регистр, FPP регистр, непосредственно значение).

Каждая команда и соответствующие ей операнды преобразуются в лексемы (слова) следующим образом:

$$Lexeme(k, n_0, n_1) = Concatenate(k, n_0, n_1),\ k \in [0, K-1],\ n_0 \in [0, N-1], n_1 \in [0, N-1],$$

где k – порядковый номер функциональной группы, к которой относится рассматриваемая команда процессора, n_0 и n_1 – порядковые номера групп, к которым относятся соответственно первый и второй операнд при данной команде.

Например, для команды «mov rax, 1»: $k = 0$ (Data Transfer Instructions), $n_0 = 1$ (General Register), $n_1 = 5$ (Immediate):

$$Lexeme(0,1,5) = '000105'$$

Таким образом, каждая функция может быть представлена в виде упорядоченного множества лексем.

4. Формирование промежуточного описания функции

Для получения векторного представления полученных лексем воспользуемся моделью word2vec [7]. Получаемые данной моделью векторные представления основываются на контекстной близости лексем внутри функций. Для обучения данной модели было использовано десяток различных бинарных файлов. В результате обучения данной модели получаем словарь, который в соответствие каждой лексеме (из присутствующих в бинарных файлах обучения) ставит вектор заданной размерности $s = 100$.

Таким образом, функция, состоящая из m команд, может быть описана в виде двумерного вектора I размерности $m \times s$. Назовем полученное описание функции промежуточным.

5. Формирование окончательного описания функции

Для построения окончательного описания функций предлагается использовать сиамскую нейронную сеть [8]. Данная сеть состоит из двух нейронных сетей с долгой краткосрочной памятью (LSTM), имеющих общие весовые коэффициенты.

Сеть LSTM представляет собой особый тип рекуррентной нейронной сети (RNN). В отличие от RNN, сеть LSTM способна работать с долговременными зависимостями. Это достигается за счет ее способности передавать состояние ячейки с предыдущего временного шага на следующий шаг, а также управлять информационным потоком в ячейке LSTM.

На вход каждой из LSTM сетей, входящих в состав сиамской сети, подаются две функции в виде полученных ранее промежуточных описаний. Затем выходы последних слоев каждой из этих LSTM сетей подаются на вход функции, оценивающей близость между полученными выходами. В данной работе в качестве такой функции используется следующая метрика:

$$D(\overline{a}, \overline{b}) = \exp(-\sum_{i=0}^{J-1} |a_i - b_i|), \quad (1)$$

где a_i – значение i-ой компоненты выходного значения последнего слоя первой сети, b_i – значение выходного значения последнего слоя второй сети. При $D = 1$ функции считаются одинаковыми, а при $D = 0$ – полностью различными.

Архитектура нейронной сети представлена на рисунке 1.

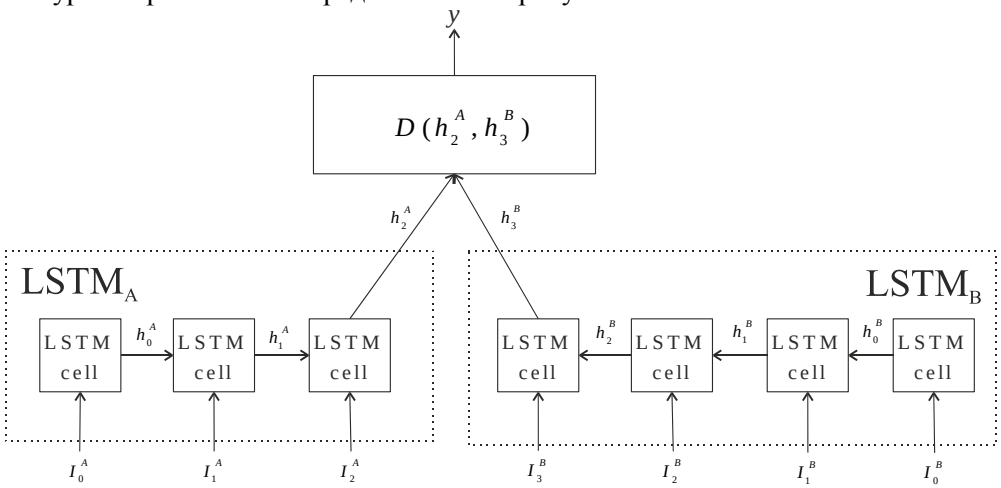

Рисунок 1. Архитектура Сиамской нейронной сети долгой краткосрочной памяти.

Здесь D – метрика (1), I_0^A, I_1^A, I_2^A – компоненты вектора промежуточного описания первой функции, соответствующие ее лексемам, $I_0^B, I_1^B, I_2^B, I_3^B$ – компоненты вектора промежуточного описания второй функции, соответствующие ее лексемам, y – выходное значение сиамской нейронной сети, принимающее значение 0, если функции разные, и 1 – если функции схожи.

В качестве функции потерь используется contrastive loss [9], имеющая следующий вид:

$$L(y, D) = \frac{1}{2}(1 - y)D^2 + \frac{1}{2}y\{\max(0, 1 - D)\}^2.$$

При указанной функции потерь и выбранной метрике (1), для одинаковых функций $(y = 1)$ нейронная сеть будет стремиться увеличить значение метрики (1) между выходами соответствующих им сетей LSTM, и, наоборот, для разных функций $(y = 0)$ – уменьшить.

Для обучения нейронной сети были использованы два «архива» функций: первый содержал функции библиотек libtiff 4.0.3 [10] и proj 4.9.1 [11], а второй функции библиотек libtiff 4.0.8 и proj 5.0.1.

В процессе обучения для каждой функции первого архива формировалось две пары: данная функция и функция, имеющая такое же название из второго архива; данная функция и случайная функция из второго архива, имеющая отличное от первой функции название. Полученные пары векторов подаются на вход нейронной сети, в качестве выходного значения для первой пары используется значение 1 (схожие), а для второй пары – 0 (различные).

Для формирования окончательного описания функций используется обученная модель сиамской нейронной сети долгой краткосрочной памяти. Подавая на два входа данной сети промежуточное описание анализируемой функции, мы можем получить значения выходного слоя одной из двух LSTM сетей – окончательное описание функции. В данной работе размерность выходного слоя $J = 32$.

Таким образом, любая функция анализируемого бинарного файла может быть представлена в виде вектора размерности J.

6. Поиск похожих функций

Получив окончательное описание функций архивных данных и текущей библиотеки, переходим к заключительному этапу представленного метода – непосредственно поиску похожих функций.

Пусть $\bar{a}$ – вектор окончательного описания функции текущей библиотеки, $\bar{b}$ – вектор окончательного описания функции библиотеки архивных данных. Для сравнения векторов признаков воспользуемся евклидовой метрикой:

$$d(\bar{a}, \bar{b}) = \sqrt{\sum_{i=0}^{J-1} (a_i - b_i)^2}, \quad (2)$$

где a_i – значение i-ой компоненты вектора признаков функции текущей библиотеки, b_i – значение i-ой компоненты вектора признаков функции библиотеки архивных данных. При $d = 0$ функции считаются одинаковыми.

Сравнив окончательное описание заданной функции текущей библиотеки с каждой функцией архивных данных, отсортируем полученные результат по увеличению расстояния (2). В результате для рассматриваемой функции текущей библиотеки получаем упорядоченный по уменьшению схожести список функций библиотеки архивных данных.

7. 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 описанную в шестом разделе метрику, для заданной функции текущей библиотеки получим упорядоченный по уменьшению схожести список функций архивных данных. Поставим в соответствие этому списку бинарную последовательность $\beta = (\beta_1, \beta_2, \ldots, \beta_L)$, *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[13,14]:

- 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*:* $AveP = \sum_{k=1}^{L} P_k (R_k - R_{k-1}), R_0 = 0$.

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

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

где S – количество функций в текущей библиотеке.

При проведении экспериментальных исследований архивные данные были представлены функциями библиотеки curl [12] версии 7.6.3, а функции текущей библиотеки – функциями различных других версий библиотеки curl.

Было проведено сравнение средней точности поиска представленного в данной работе метода поиска и известных ранее методов: метод, основанный на анализе пространственного положения функциональных групп команд процессора [15] и метод, основанный на сравнении k-грамм [2]. 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 1.

Таблица 1. 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 [13] метод
libcurl 7.5.4	0.7830	**0.8208**	0.7828
libcurl 7.5.6	**0.8631**	0.8543	0.8550
libcurl 7.5.9	**0.8886**	0.8768	0.8851
libcurl 7.6.0	0.8960	0.8830	**0.9036**

Анализ полученных результатов показывает, что представленный в данной работ метод поиска похожих последовательностей кода не уступает известным методам, а в некоторых случаях и превосходит их. Так как обучение сиамской нейронной сети выполнялось на небольшом наборе данных, полученные для данного метода результаты в дальнейшем могут быть улучшены, путем увеличения объема данных для обучения нейронной сети.

8. Заключение

В работе представлен метод поиска похожих последовательностей кода в исполняемых файлах на основе сиамской нейронной сети. Представлены результаты экспериментальных исследований, демонстрирующие работоспособность и эффективность разработанного метода (средняя точность поиска 0.78 - 0.89). Дальнейшие исследования будут направлены на настройку параметров сиамской нейронной сети с целью увеличения средней точности поиска данного метода.

9. Благодарности
Работа выполнена при финансовой поддержке гранта РФФИ № 18-01-00748 А.

10. Литература

[1] An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples [Electronic resource]. – Access mode: https://arxiv.org/pdf/1910.01321 (18.11.2019).
[2] Myles, G. K-gram based software birthmarks / G. Myles, C. Collberg // Proceedings of the ACM symposium on Applied computing. – 2005. – P. 314-318. DOI: 10.1145/1066677. 1066753.
[3] 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 (18.11.2019).
[4] Shalev, M. Binary Similarity Detection Using Machine Learning / M. Shalev, N. Partush // Proceedings of the 13th Workshop on Programming Languages and Analysis for Security. – 2018. – P. 42-47. DOI: 10.1145/3264820.3264821.
[5] 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.
[6] Hex-Rays IDA: About [Electronic resource]. – Access mode: http://hex-rays.com/products/ida/ (18.11.2019).
[7] Efficient Estimation of Word Representations in Vector Space [Electronic resource]. – Access mode: https://arxiv.org/pdf/1301.3781 (18.11.2019).
[8] Bromley, J. Signature verification using a "Siamese" time delay neural network / J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah // Proceedings of the 6th International Conference on Neural Information Processing Systems. – 1994. – P. 737-744.
[9] Hadsell, R. Dimensionality Reduction by Learning an Invariant Mapping / R. Hadsell, S. Chopra, Y. LeCun // IEEE Computer Society – 2006. – Vol. 2. – P. 1735-1742. DOI: 10.1109/ CVPR.2006.100.
[10] TIFF Library and Utilities [Electronic resource]. – Access mode: http://www.libtiff.org/ (18.11.2019).
[11] PROJ coordinate transformation software library [Electronic resource]. – Access mode: https://proj.org/about.html (18.11.2019).
[12] Libcurl - the multiprotocol file transfer library [Electronic resource]. – Access mode: https://curl.haxx.se/libcurl/ (18.11.2019).
[13] 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.
[14] 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(1). – P. 37-63.
[15] 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.

Searching for similar code sequences in executable files using siamese neural network

A.S. Yumaganov[1]

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

Abstract. This work is dedicated to solving the problem of finding similar code sequences (functions) in executable files. The description of functions obtained using the proposed method for solving this problem is based on the mutual spatial position of processor instructions and the corresponding operands in the function body. The word embedding model word2vec is used to form an intermediate description of the executable file functions. The final description of the functions is formed using the siamese long short-term memory network (Siamese-LSTM). Then it description directly used to search for similar functions. The results of experimental studies of the developed method are presented in comparison with some previously known methods.