Платформенно-независимый и масштабируемый инструмент поиска клонов кода в бинарных файлах*

А.К. Асланян <hayk@ispras.ru>
Ш.Ф. Курмангалеев <kursh@ispras.ru>
В.Г. Варданян <vaag@ispras.ru>
М.С. Арутюнян <arutunian@ispras.ru>
С.С. Саргсян <sevaksargsyan@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. При разработке программного обеспечения разработчики часто прибегают к копированию того или иного участка кода для достижения желаемого результата. Копирование кода может привести к появлению различных ошибок, а также к увеличению размера исходного и бинарного кода. Задача поиска семантически сходных участков кода (клонов) в бинарных файлах становится более актуальной в связи с недоступностью исходного кода многих программных средств. В данной статье обсуждаются существующие методы поиска клонов бинарного кода и приводится описание разработанного нами инструмента обнаружения клонов в бинарном коде. Работа инструмента разделена на три основных этапа. Первый этап базируется на платформе Binnavi [1] и ответственен за генерацию графов зависимостей программы для каждой функции. В качестве основы для генерации графов используется платформенно-независимый язык REIL (Reverse Engineering Intermediate Language). Использование языка REIL позволяет генерировать графы сразу для нескольких целевых архитектур (x86, x86-64, ARM, MIPD, PPC), тем самым обеспечивает независимость инструмента от целевой архитектуры. На втором этапе производится поиск клонов на основе ранее созданных графов. Для каждой пары графов строится наибольший общий подграф, на основе которого определяются клоны бинарного кода. На третьем этапе полученные клоны визуализируются для удобного анализа полученных результатов.

Ключевые слова: клоны кода; семантический анализ бинарного кода; REIL; граф зависимостей программы.

DOI: 10.15514/ISPRAS-2016-28(5)-13

* Работа поддержана грантом РФФИ № 15-07-07541 А

Для цитирования: А.К. Асланян, Ш.Ф. Курмангалеев, В.Г. Варданян, М.С. Арутюнян, С.С. Саргсян. Платформенно-независимый и масштабируемый иструмент поиска клонов бинарного кода. Труды ИСП РАН, том 28, вып. 5, стр. 215-226, 2016 г. DOI: 10.15514/ISPRAS-2016-28(5)-13

1. Введение

Существует ряд методов поиска клонов кода, основанный на текстовом [2], лексическом [3], синтаксическом [4, 5, 6] и семантическом [6, 7, 8, 9, 10, 11, 12 13] анализе программы. Но, в основном, все эти методы основаны требуют наличия исходного кода программы. Задача поиска клонов в бинарном коде мало изучена несмотря на то, что она является более важной с точки зрения поиска ошибок в программах, учитывая тот факт, что в основном программы распространяются без исходного кода. Качественный инструмент поиска клонов бинарного кода может быть применен в задачах поиска необновленных фрагментов и вирусов в исполняемых файлах.

Клоны бинарного кода условно разделены на три основные типы. Первый тип – фрагменты кода, которые полностью совпадают. Второй тип – фрагменты кода, которые могут отличаться типами, значениями данных именами регистров. Третий тип – фрагменты кода, которые могут отличаться типами, значениями данных именами регистров, а также могут отличаться некоторыми инструкциями (в конкретном фрагменте могут присутствовать или отсутствовать некоторые инструкции).

В рис. 1 приведены примеры клонов кода в ассемблерной форме (для архитектуры x86). Клон первого типа совпадает с конкретным фрагментом. Клон второго типа от конкретного фрагмента отличается распределением регистра *ecx* в место *eax*. Клон третьего типа от конкретного фрагмента отличается распределением регистра *ecx* в место *eax* и отсутствием одной инструкции (*imul eax, ebp+var_4]*).

2. Подходы поиска клонов в бинарном коде

Существует несколько работ, посвященные поиску клонов в бинарном коде. Алгоритм, названный BitShred [14], основан на «отпечатках» фрагментов, использует фильтр для поиска ошибок, появившихся в результате дублирования кода. Фильтр – структура данных, позволяющая проводить проверку принадлежности элемента набору отпечатков. Алгоритм BitShred состоит из 3 этапов: разбиения файла, создания «отпечатков» для полученных фрагментов и сравнения этих «отпечатков» между собой. Для «отпечатков» считаются хеш коды и если они совпадают, то соответствующие им фрагменты считаются клонами. Алгоритм находит клоны только первого типа.

Фрагмент кода	Клон типа 1	Клон типа 2	Клон типа 3

public main
main proc near
 	
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
 	
push ebp
mov ebp, esp
mov [ebp+var_4], 5
mov eax,[ebp+var_4]
imul eax,[ebp+var_4]
leave
retn
main endp
	public main
	main proc near
	
	var_4= dword ptr -4
	argc= dword ptr 8
	argv= dword ptr 0Ch
	envp= dword ptr 10h
	
	push ebp
	mov ebp, esp
	mov [ebp+var_4], 5
	mov eax,[ebp+var_4]
	imul eax,[ebp+var_4]
	leave
	retn
	main endp
		public main
		main proc near
		
		var_4= dword ptr -4
		argc= dword ptr 8
		argv= dword ptr 0Ch
		envp= dword ptr 10h
		
		push ebp
		mov ebp, esp
		mov [ebp+var_4], 10
		mov ecx, [ebp+var_4]
		imul ecx, [ebp+var_4]
		leave
		retn
		main endp
			public main
			main proc near
			
			var_1= dword ptr -4
			argc= dword ptr 8
			argv= dword ptr 0Ch
			envp= dword ptr 10h
			
			push ebp
			mov ebp, esp
			mov [ebp+var_1], 15
			mov ecx, [ebp+var_1]
			
			leave
			retn
			main endp

Рис. 1. Примеры типов клонов для архитектуры x86 (соответствующий ассемблер)

Fig. 1. Examples of clones types for x86 architecture (corresponding assembler)

A. Schulman [15] предложил систему, которая создаёт хеш для каждой функции в бинарном файле. Совпадающие хеши, встретившиеся более чем в одном файле указывают на клон кода. Алгоритм позволяет находить клоны только первого типа, так как каждый раз компилятор может разместить регистры по-разному.

Система, созданная D. Bruschi и др. [16], находит клоны в бинарных файлах для обнаружения вредоносных программ. Сначала бинарный файл дизассемблируется и нормализуется, после чего удаляется мертвый код и происходит разделение кода на фрагменты. Для каждого фрагмента строится метрика, основанная на графе потока управления. На последнем этапе происходит поиск клонов, при помощи сравнения полученных метрик.

Sæbjørnsen и др. [17] после получения ассемблера из бинарного файла, нормализируют и считают характеризующие векторы. На основе сравнивания этих векторов становится возможным нахождение клонов типов T1, T2 и T3. На основе этой работы, M. Farhadi и др. [18] создали систему для обнаружения клонов вредоносного кода в программах.

T. Dullien и др. [19] предложили систему сравнения бинарных файлов на основе структурного анализа, для поиска вредоносного кода. Алгоритм состоит из двух этапов: генерация признаков вредоносного кода и распознавание схожести между различными участками кода на основе графа потока управления.

3. Модель инструмента поиска клонов в бинарных файлах

Предлагаемый модель инструмента для поиска клонов бинарного кода был разработан с учетом следующих требований:

- нахождение T1, T2 и T3 типов клонов;
- независимость от целевой архитектуры;
- масштабируемость: размер анализируемых программ может достигать десятков МБ;
- большой процент истинных срабатываний (> 90%).

При анализе кода, учитываются параметры, задаваемые пользователем: минимальное число инструкций для конечных клонов (МЧИ) и минимальный процент схожести (МПС) клонов.

Работа инструмента делится на три основных этапа:

- Первый этап базируется на платформе Binnavi (инфраструктура для анализа бинарных файлов. В качестве инструмента для восстановления структур и потока управления программы используется дизассемблер Ida Pro [20]). На этом этапе происходит трансляция из машинного кода в представление REIL из которого, в свою очередь, генерируются ГЗП (граф зависимостей программы) для каждой функции. Узлам ГЗП соответствуют инструкции REIL, а ребрам – зависимости по данным и по управлению. В конце первого этапа все графы сериализуются.
- На втором этапе производится поиск клонов ассемблерного кода, учитывая параметры МЧИ и МПС.
- На третьем этапе демонстрируются полученные клоны и соответствующие им графы.

Промежуточная сериализация и десериализация дает два основных преимущества. Первое – для некоторых проектов работа с созданными графами может потребовать большое количество оперативной памяти. Сериализация графов позволяет извлечь и сравнивать графы парами, тем самым сэкономив память. Второе - возможность многократного применения второго этапа с различными значениями для параметров МЧИ и МПС.

На рис.2 приведена схема работы инструмента:

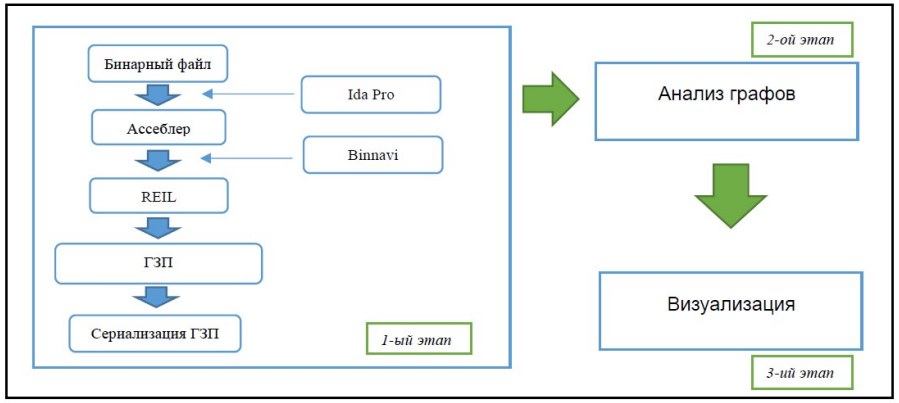

Рис. 2. Схема работы инструмента

Fig. 2. The tool architecture

3.1 Генерация ГЗП

Для генерации графов зависимостей программы используются средства и возможности платформы Binnavi. Binnavi предоставляет интерфейс для генерации и использования разных промежуточных представлений программы, основанных на языке REIL, в том числе, генерация графа потока управления, генерация графа вызовов функций и графа зависимостей по данным. В рамках разработки инструмента в платформу Binnavi был добавлен новый функционал, который позволяет для каждой функции автоматически генерировать граф потока управления и граф зависимостей по данным и объединить их в единый граф зависимостей программы (рис. 3). После чего, созданные графы сохраняются для дальнейшего анализа.

3.2 Разделение ГЗП на подграфы

После загрузки ГЗП, они могут быть разделены на единицы сравнения (ЕС). ЕС-ы представляют собой подграфы ГЗП и рассматриваются как потенциальные клоны друг друга. Так как графы изначально генерируются для функций, то для сравнения кода в рамках каждой функции необходимо разделение графов. Разработаны два метода для решения этой задачи: разделение, учитывая базовые блоки кода и разделение по слабо связанным компонентам графа. Разделение графов также позволяет увеличить эффективность алгоритмов анализа ГЗП (алгоритмы поиска максимальных изоморфных подграфов)

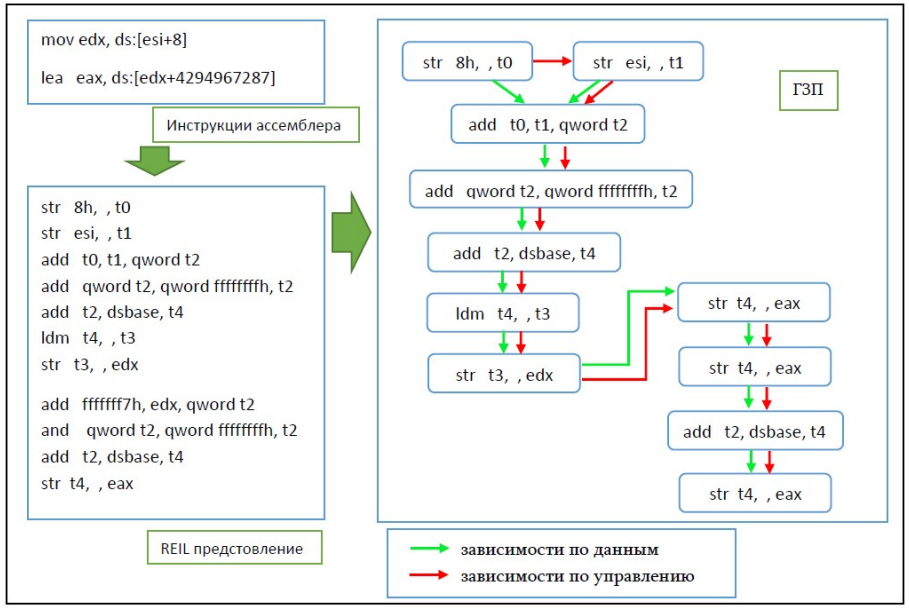

Рис. 3. Пример генерации ГЗП

Fig. 3. Example of PDG generation

3.3 Анализ ГЗП графов

На этом шагу происходит анализ ЕС для нахождения клонов кода. После загрузки ЕС они попарно сравниваются. Для каждой пары строится наибольший общий подграф. Как известно, эта проблема в общем случае NP-полная и для решения проблемы используется приближенный алгоритм. В большинстве случаев созданные графы разреженные, что и используется для более эффективного нахождения наибольшего общего подграфа.

На первом этапе алгоритма производится фильтрация пар ГЗП по параметру МЧИ. Сложность этого этапа составляет $O(n \cdot \log_2 n)$, где n - количество вершин в обеих ЕС. С помощью этого этапа эффективность алгоритма значительно повышается. Вторая часть алгоритма находит наибольший общий подграф и проверяет удовлетворение параметра МПС. Сначала сопоставляются те вершины в графах, которые не имеют входных ребер. В следующих итерациях рекурсивно рассматриваются и сопоставляются вершины связаные ребром с вершинами, которые в предыдущих итерациях сопоставлялись. Сложность этой части алгоритма не превышает $O(n^3)$, где n - количество вершин в обеих ЕС. После нахождения наибольшего общего подграфа, восстанавливаются фрагменты.

3.4 Фильтрация полученных клонов

Так как ГЗП строится из представления REIL, в некоторых случаях полученные машинные инструкции, соответствующие изоморфным подграфам, могут быть сильно разбросаны и не удовлетворять условию МПС. По этой причине возникает дополнительная необходимость фильтрации полученных клонов.

3.5 Визуализация полученных клонов

Последний этап работы инструмента – визуализация клонов. Цель созданного графического интерфейса - демонстрация ассемблерного кода полученных клонов, процент их схожести, а также соответствующие им графы (рис. 4).

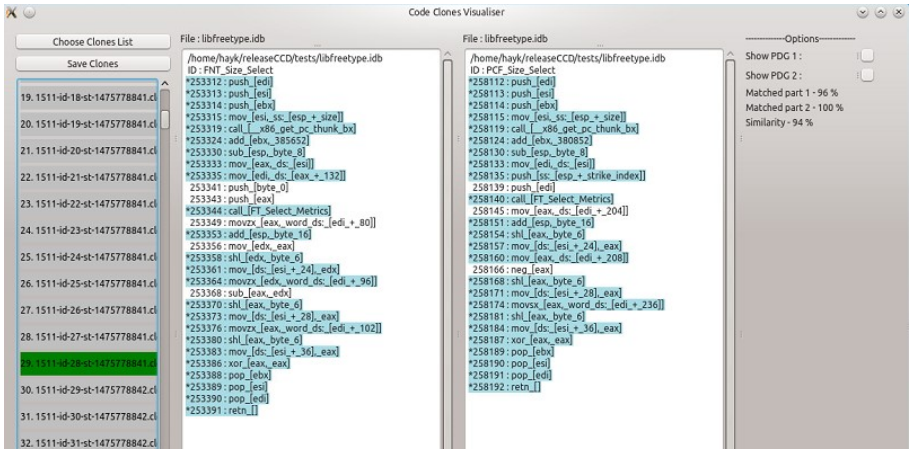

Рис. 4. Демонстрация результатов

Fig. 4. Visualization of results

4. Результаты

Для оценки эффективности инструмента произведено тестирование на разных тестовых и реальных проектах. Целевая машина – Intel Xeon, 40 ядер, ОЗУ - 128 ГБ.

Результаты приведены на таб. 1 (бинарные файлы получены компиляцией исходного кода с флагами -O0 –fno-inline) и таб. 2 (бинарные файлы получены компиляцией исходного кода с флагами –O3 –fno-inline), где МЧИ равно 30, МПС равно 90%, сравнивались функции для каждого бинарного файла между собой.

Табл. 1. Результаты тестирования

Table 1. Results

Бинарный файл	Размер бинарного файла	Количество найденных клонов	Время работы инструмента
libxml2.so	1.8 МБ	2123	1м.46 с.
libfreetype.so	816 КБ	270	57 с.
openssl	764 КБ	40	35 с.
d8	33 МБ	40397	26 м.52 с.

Табл. 2. Результаты тестирования

Table 2. Results

Бинарный файл	Размер бинарного файла	Количество найденных клонов	Время работы инструмента
libxml2.so	1.8 МБ	437	1м.25 с.
libfreetype.so	740 КБ	73	36 с.
openssl	672 КБ	71	26 с.
d8	20 МБ	19453	17 м.26 с.

5. Дальнейшая работа

Планируется применить инструмент для поисков фрагментов кода с ошибками и вредоносным кодом. При наличии базы вирусов или фрагментов кода, которые содержат ошибки, инструмент может найти соответствующие им клоны, которые с большой вероятностью тоже содержат ошибки или вредоносный код. Инструмент также может применяться для автоматического поиска однотипных ошибок.

6. Заключение

В данной работе рассмотрены существующие подходы поиска клонов для бинарного кода. Разработан платформенно-независимый, масштабируемый инструмент, который позволяет находить все три типа клонов кода, и обладает

высокой точностью. Результаты тестирования показали, что все критерии, поставленные нами, удовлетворяются.

Список литературы

[1]. https://www.zynamics.com/binnavi.html

[2]. Ducasse S., Rieger M., Demeyer S., A language independent approach for detecting duplicated code, in: Proceedings of the 15th International Conference on Software Maintenance, 1999, pp. 109-119, DOI: 10.1109/ICSM.1999.792593.

[3]. Kamiya T., Kusumoto S., Inoue K., CCFinder: A multilinguistic tokenbased code clone detection system for large scale source code, IEEE Transactions on Software Engineering, 2002, vol. 28, no. 7, pp. 654-670, DOI: 10.1109/TSE.2002.1019480.

[4]. Baxter I., Yahin A., Moura L., Anna M., Clone detection using abstract syntax trees, in: Proceedings of the 14th IEEE International Conference on Software Maintenance, IEEE Computer Society, 1998, pp. 368-377, DOI: 10.1109/ICSM.1998.738528.

[5]. Tairas R., Gray J., Phoenix-based clone detection using suffix trees, in: Proceedings of the 44th Annual Southeast Regional Conference, 2006, pp. 679-684, DOI: 10.1145/1185448.1185597.

[6]. Jiang L., Misherghi G., Su Z., Glondu S., DECKARD : Scalable and accurate tree-based detection of code clones, in: Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 2007, pp. 96-105, DOI: 10.1109/ICSE.2007.30.

[7]. Komondoor R., Horwitz S., Using slicing to identify duplication in source code, in: Proceedings of the 8th International Symposium on Static Analysis, 2001, pp. 40-56, DOI: 10.1007/3-540-47764-0_3.

[8]. Krinke J., Identifying similar code with program dependence graphs, in: Proceedings of the 8th Working Conference on Reverse Engineering, 2001, pp. 301-309, DOI: 10.1109/WCRE.2001.957835.

[9]. Gabel M., Jiang L., Su Z., Scalable detection of semantic clones, in: Proceedings of 30th International Conference on Software Engineering, 2008, pp. 321-330, DOI: 10.1145/1368088.1368132.

[10]. Sargsyan S., Kurmangaleev S., Baloian A.., Aslanyan H., Scalable and Accurate Clones Detection Based on Metrics for Dependence Graph, Mathematical Problems of Computer Science, Volume 42, 2014, pp. 54-62.

[11]. Avetisyan A., Kurmangaleev S., Sargsyan S., Arutunian M., Belevantsev A. LLVMBased Code Clone Detection Framework. 10th International Conference on Computer Science and Information Technologies, 2015, pp. 178-182.

[12]. Саргсян С., Курмангалеев Ш., Белеванцев А., Аветисян А. Масштабируемый и точный поиск клонов кода. Программирование, № 6, 2015, стр. 9-17.

[13]. Саргсян С., Курмангалеев Ш., Белеванцев А., Асланян А., Балоян А. Масштабируемый инструмент поиска клонов кода на основе семантического анализа программ. Труды ИСП РАН, том 27, вып. 1, 2016, стр. 39-50. DOI: 10.15514/ISPRAS-2015-27(1)-3

[14]. J. Jang and D. Brumley. Bitshred: Fast, scalable code reuse detection in binary code (cmu-cylab-10-006). CyLab, 2009 page 28.

[15]. A. Schulman. Finding binary clones with opstrings function digests: Part III. Dr. Dobb's Journal, 30(9):64, 2005.

[16]. D. Bruschi, L. Martignoni, and M. Monga. Code normalization for self-mutating malware. IEEE Security & Privacy, 5(2):46–54, 2007.

[17]. A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su. Detecting code clones in binary executables. In Proceedings of the 18th International Symposium on Software Testing and Analysis, ACM, 2009, pp. 117–128.

[18]. M. R. Farhadi, B. C. M. Fung, P. Charland and M. Debbabi, "BinClone: Detecting Code Clones in Malware," Software Security and Reliability (SERE), 2014 Eighth International Conference on, San Francisco, CA, 2014, pp. 78-87. doi: 10.1109/SERE.2014.21.

[19]. Thomas Dullien, Ero Carrera, Soeren-Meyer Eppler, Sebastian Porst «Automated attacker correlation for malicious code» // Technical report, DTIC Document, 2010.

[20]. https://www.hex-rays.com/products/ida/

Platform-independent and scalable tool for binary code clone detection★

[1] *H.K. Aslanyan <hayk@ispras.ru>*

[1] *S.F. Kurmangaleev <kursh@ispras.ru>*

[1] *V.G. Vardanyan <vaag@ispras.ru>*

[1] *M.S. Arutunian <aritunian@ispras.ru>*

[1] *S.S. Sargsyan <sevaksargsyan@ispras.ru>*

[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. During the software development developers often copy and paste fragments of code to achieve the desired result. Copying of code can lead to variety of errors, as well as can increase the size of the source and binary code. The problem of finding semantically similar pieces of code (clones) in binary code becomes actual due to the unavailability of source code of many software programs. The first part of the article is dedicated to the analysis of the existing methods for finding code clone in binary code. In the second part we provide a newly developed tool for finding code clones in binary code. The work of the tool is divided into three main stages. The first stage is based on the Binnavi [1] framework, which is responsible for generation of program dependence graphs (PDG). Program dependence graphs are generated using REIL (Reverse Engineering Intermediate Language). The usage of REIL language allows to generate graphs for multiple architectures (x86, x86-64, ARM, MIPS, PPC), thus providing the independence of the tool from the target architecture. In the second step code clones are found based on previously created graphs. Maximum common subgraph is built for each pair of graphs and based on it, code clones are detected. In the third stage, the detected clones are visualized for convenient analysis of the results.

Keywords: code clone; semantic analysis of binary code; REIL; program dependence graph.

★ The paper is supported by RFBR grant 15-07-07541 A

DOI: 10.15514/ISPRAS-2016-28(5)-13

For citation: H.K. Aslanyan, S.F. Kurmangaleev, V.G. Vardanyan, M.S. Arutunian, S.S. Sargsyan. Platform-independent and scalable tool for binary code clone detection. *Trudy ISP RAN/Proc. ISP RAS*, vol. 1, issue 2, 2016. pp. 215-226 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-13

References

[1]. https://www.zynamics.com/binnavi.html
[2]. Ducasse S., Rieger M., Demeyer S., A language independent approach for detecting duplicated code, in: Proceedings of the 15th International Conference on Software Maintenance, 1999, pp. 109-119, DOI: 10.1109/ICSM.1999.792593.
[3]. Kamiya T., Kusumoto S., Inoue K., CCFinder: A multilinguistic tokenbased code clone detection system for large scale source code, IEEE Transactions on Software Engineering, 2002, vol. 28, no. 7, pp. 654-670, DOI: 10.1109/TSE.2002.1019480.
[4]. Baxter I., Yahin A., Moura L., Anna M., Clone detection using abstract syntax trees, in: Proceedings of the 14th IEEE International Conference on Software Maintenance, IEEE Computer Society, 1998, pp. 368-377, DOI: 10.1109/ICSM.1998.738528.
[5]. Tairas R., Gray J., Phoenix-based clone detection using suffix trees, in: Proceedings of the 44th Annual Southeast Regional Conference, 2006, pp. 679-684, DOI: 10.1145/1185448.1185597.
[6]. Jiang L., Misherghi G., Su Z., Glondu S., DECKARD : Scalable and accurate tree-based detection of code clones, in: Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 2007, pp. 96-105, DOI: 10.1109/ICSE.2007.30.
[7]. Komondoor R., Horwitz S., Using slicing to identify duplication in source code, in: Proceedings of the 8th International Symposium on Static Analysis, 2001, pp. 40-56, DOI: 10.1007/3-540-47764-0_3.
[8]. Krinke J., Identifying similar code with program dependence graphs, in: Proceedings of the 8th Working Conference on Reverse Engineering, 2001, pp. 301-309, DOI: 10.1109/WCRE.2001.957835.
[9]. Gabel M., Jiang L., Su Z., Scalable detection of semantic clones, in: Proceedings of 30th International Conference on Software Engineering, 2008, pp. 321-330, DOI: 10.1145/1368088.1368132.
[10]. Sargsyan S., Kurmangaleev S., Baloian A.., Aslanyan H., Scalable and Accurate Clones Detection Based on Metrics for Dependence Graph, Mathematical Problems of Computer Science, Volume 42, 2014, pp. 54-62.
[11]. Avetisyan A., Kurmangaleev S., Sargsyan S., Arutunian M., Belevantsev A. LLVMBased Code Clone Detection Framework. 10th International Conference on Computer Science and Information Technologies, 2015, pp. 178-182.
[12]. Sargsyan S., Kurmangaleev S., Belevantsev A., Avetisyan A. Scalable and accurate detection of code clones. Programming and Computer Software, 2016, issue 1. pp. 27-33. DOI: 10.1134/S0361768816010072
[13]. Sargsyan S., Kurmangaleev S., Belevantsev A., Aslanyan H., Baloian A. [Scalable tool for code clone detection based on semantic analysis of program]. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 1, pp. 39-50 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-3

[14]. J. Jang and D. Brumley. Bitshred: Fast, scalable code reuse detection in binary code (cmu-cylab-10-006). CyLab, 2009 page 28.
[15]. A. Schulman. Finding binary clones with opstrings function digests: Part III. Dr. Dobb's Journal, 30(9):64, 2005.
[16]. D. Bruschi, L. Martignoni, and M. Monga. Code normalization for self-mutating malware. IEEE Security & Privacy, 5(2):46–54, 2007.
[17]. A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su. Detecting code clones in binary executables. In Proceedings of the 18th International Symposium on Software Testing and Analysis, ACM, 2009, pp. 117–128.
[18]. M. R. Farhadi, B. C. M. Fung, P. Charland and M. Debbabi, "BinClone: Detecting Code Clones in Malware," Software Security and Reliability (SERE), 2014 Eighth International Conference on, San Francisco, CA, 2014, pp. 78-87. doi: 10.1109/SERE.2014.21.
[19]. Thomas Dullien, Ero Carrera, Soeren-Meyer Eppler, Sebastian Porst «Automated attacker correlation for malicious code» // Technical report, DTIC Document, 2010.
[20]. https://www.hex-rays.com/products/ida/