13. Intel Xeon Phi Coprocessor System Software Developers Guide. Intel Corporation, 2014, 164 p.
14. *Al'bert'yan A.M., Kurochkin I.I.* Ispol'zovanie soprotsessorov Intel Xeon Phi v grid-sistemakh iz personal'nykh komp'yuterov [The use of Intel Xeon Phi coprocessors in grid systems from personal computers], *CEUR-Proceedings: Selected Papers of the II Intern. Sci. Conf." Convergent Cognitive Information Technologies", Moscow, Russia*, 2017, Vol. 2064, pp. 196-201.
15. *Vatutin E., Belyshev A., Nikitina N., Manzuk M.* Evaluation of Efficiency of Using Simple Transformations When Searching for Orthogonal Diagonal Latin Squares of Order 10, *Communications in Computer and Information Science*, Vol. 1304. Springer, 2020, pp. 127-146. DOI: 10.1007/978-3-030-66895-2_9.
16. *James Jeffers, James Reinders* Intel Xeon Phi Processor High Performance Programming. Morgan Kaufmann, 2013, 432 p. ISBN: 978-0-12-410414-3.
17. *De Ravé E.G. et al.* Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm, *Computers & Geosciences*, 2014, Vol. 64, pp. 1-6.
18. *Nobile M.S. et al.* Graphics processing units in bioinformatics, computational biology and systems biology, *Briefings in bioinformatics*, 2017, Vol. 18, No. 5, pp. 870-885.
19. *Morrison D.R. et al.* Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning, *Discrete Optimization*, 2016, Vol. 19, pp. 79-102.
20. *Hill M.D. et al.* On the Spectre and Meltdown processor security vulnerabilities, *IEEE Micro*, 2019, Vol. 39, No. 2, pp. 9-19.

Статью рекомендовал к опубликованию д.т.н., профессор А.И. Баранчиков.

Альбертьян Александр Михайлович – Федеральный Исследовательский Центр «Информатика и управление» РАН; e-mail: admin@isa.ru; Москва, Россия; УИТС; ведущий инженер.

Курочкин Илья Ильич – Институт проблем передачи информации им. А.А. Харкевича РАН; e-mail: kurochkin@iitp.ru; Москва, Россия; тел.: +74956504225; Центр распределенных вычислений, лаборатория Ц-1; зав. лабораторией; к.т.н.

Ватутин Эдуард Игоревич – Юго-Западный государственный университет; e-mail: evatutin@rambler.ru; г. Курск, Россия; тел.: +74712222670; кафедра вычислительной техники; к.т.н.; доцент.

Albertian Alexander Mikhaylovich – Federal Research Center "Computer Science and Control", Russian Academy of Sciences; e-mail: admin@isa.ru; Moscow, Russia; IT department.

Kurochkin Ilya Il'ich – Institute for Information Transmission Problems, Russian Academy of Sciences; e-mail: kurochkin@iitp.ru; Moscow, Russia; phone: +74956504225; Distributed computing center, Lab-C1, cand. of eng. sc.

Vatutin Eduard Igorevich – Southwest State University; e-mail: evatutin@rambler.ru; Kursk, Russia; phone: 84712222670; the department of computer science; cand. of eng. sc.; associate professor.

УДК 004.056 DOI 10.18522/2311-3103-2021-7-153-167

Л.К. Бабенко, А.С. Кириллов

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ОБНАРУЖЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

При проведении исследований в области обнаружения вредоносного программного обеспечения основной фокус делается именно на методах, игнорируя то, как эти методы практически могли бы быть реализованы. С другой стороны, есть работы, которые раскрывают некоторые технические подробности реализации или оптимизации процесса анализа исследуемого образца и сбора данных о его работе. Однако, необходимо соединять результаты концепций экспериментальных систем и те возможности реализации, кото-

рые имеются. Целью работы является описание реализации автоматизированной системы обнаружения вредоносного программного обеспечения на основе предложенного ранее авторами метода, таким образом, дополняя результаты прошлых исследований и реализуя на практике предложенный метод обнаружения и кластеризации вредоносного программного обеспечения. В результате, раскрыты технические требования к проектируемой системе обнаружения вредоносного программного обеспечения, обусловленные предложенным ранее методом обнаружения и кластеризации. Произведено сравнение существующих средств поведенческого анализа, в качестве наиболее подходящей выбрана Cuckoo Sandbox, основным ее достоинством является открытость исходных текстов, что обеспечило возможность доработки как ее клиентской части, так и серверной части. В частности выполнено расширение списка контролируемых системных функций, определение модуля-источника вызова, определение контекста вызова. Так же, на основе Cuckoo Sandbox было разработано расширение, которое реализует предложенный авторами метод. Далее в статье раскрывается возможность портирования описанной системы для работы с образцами вредоносного программного обеспечения, разработанными под различные платформы. В частности, показано, что предложенные методы, могут быть адаптированы под такие платформы как .NET или Android, при этом доработки носят технический, а не принципиальных характер. С практический точки зрения, система представляет из себя программный комплекс для специалиста по безопасности и позволяет осуществлять оперативное обнаружение неизвестных ранее угроз и вместе с тем, за счет проведения кластеризации, идентифицировать конкретную угрозу для реализации наиболее подходящих мер защиты от этой угрозы. В предложенном виде, может быть использована как часть инфраструктуры предприятия для обеспечения антивирусной безопасности.

Статический анализ; поведенческий анализ; трассы системных функций; линковка, контекст вызова; исполнимый модуль; структура системы; доработка исходного кода; портирование.

L.K. Babenko, A.S. Kirillov

DEVELOPMENT OF AUTOMATED MALWARE DETECTION SYSTEM

When research in the field of malware detection, the authors focus exclusively on detection methods, ignoring how these methods could practically be implemented. On the other hand, there are works that reveal some technical details of the implementation or optimization of the process of analyzing the malware sample and collecting data on its work. However, it is necessary to combine the results of the concepts of experimental systems and the implementation possibilities that are available. The purpose of the work is description of the implementation of an automated malware detection system based on the method proposed earlier by the authors, thus supplementing the results of previous studies and putting into practice the proposed method for detecting and clustering malware. As a result, the technical requirements for the developed system for detecting malware is described, due to the previously proposed method of detection and clustering. A comparison of existing behavioral analysis tools was made, Cuckoo Sandbox was chosen as the most suitable one, its main advantage is the open source code, which made it possible to refine both its client part and server part. In particular, the list of controlled system functions has been expanded, the source module of the call has been determined, and the call context has been determined. Also, based on the Cuckoo Sandbox, an extension has been developed that implements the method proposed by the authors. The article also reveals the possibility of porting the described system to work with samples of malware developed for various platforms. In particular, it is shown that the proposed methods can be adapted to platforms such as .NET or Android, while the improvements are technical, not fundamental. From a practical point of view, the system is a software package for a security specialist and allows for the rapid detection of previously unknown threats and, at the same time, through clustering, to identify a specific threat in order to implement the most appropriate protection measures against this threat. In the proposed form, it can be used as part of the enterprise infrastructure to ensure anti-virus security.

Static analysis; dynamic analysis; system function traces; linking; call context; executable module; system structure; source code modification; porting.

Введение. Вопрос разработки методов и систем обнаружения вредоносного программного обеспечения (ВПО) не теряет актуальности. По данным компании AV-Test с 2013 года количество обнаруженных образцов ВПО увеличилось в 10 раз и в 2021 году достигло 1312 миллионов экземпляров за год. Ранее авторами уже был предложен метод обнаружения [2] и кластеризации ВПО [3], который показал хорошие результаты, однако работа над проблемой обнаружения ВПО была бы не полной, без представления системы, которая бы реализовывала этот метод. Предложенный метод преимущественно ориентирован на ОС Windows которая, с одной стороны де-факто самая распространенная ОС, с другой стороны – самая частая цель ВПО [1], по этой причине, далее все рассматривается именно в контексте этой ОС и платформы Win32/Win64. В данной работе представлены результаты по созданию автоматизированной системы обнаружения ВПО, а также раскрывается возможность портирования предложенной системы для работы с образцами ВПО, разработанными под платформы отличные от Win32/Win64. Под системой обнаружения ВПО понимается программный комплекс, выполняющий статический и поведенческий анализ набора образцов и последующую идентификацию образцов ВПО с назначением им меток семейств, согласно предложенному ранее авторами методу. Рассмотрим работы исследователей в данной области. Существуют патенты, дающие общее представление о работе таких систем, например в патенте [4] хорошо описана программная архитектура системы обнаружения ВПО, в патенте [5] раскрыт вопрос создания специальной среды для методов обнаружения ВПО, так же можно найти работы, раскрывающие преимущественно технические моменты. В работе [6] авторы описали архитектуру и реализацию аппаратного решения – песочницы для поведенческого анализа ВПО. Другим примером такой работы - оптимизация работы виртуальных сред для анализа ВПО в [7]. Большая же часть работ содержат подробное описание методов, которые лежат в основе системы и либо общее, либо не полное, схематическое описание такой системы. Например, в [8] представлена только логическая структура предложенной системы. В работе [9] представлено описание облачной системы обнаружения ВПО, основное внимание в работе уделяется программной архитектуре, по большому счету не специфичной для задачи обнаружения ВПО, а характерной для любой облачной системы обработки данных. В работе [10] дано описание логической структуры предложенной системы на основе статического анализа без детализации относительно того, как система реализована, кроме того, минимально раскрываются даже программные средства, на основе которых реализована система. В работе [11] авторы так же предложили схематическое описание предложенной системы, не представлено никаких данных по реализации предложенной системы несмотря на то, что выполнения поведенческого анализа не предполагается, а это обычно самая сложная часть. Одной из интересных работ, которая описывает создание системы обнаружения ВПО является [12], авторы предложили и описали систему, на основе закрытой системы поведенческого анализа, однако, предложенное решение хоть и обозначается как система, но фактически представляет из себя разрозненный набор программных средств, не интегрированных в общую инфраструктуру. Конечно, нельзя говорить, что вопрос создания автоматизированных систем обнаружения ВПО совсем не раскрыт. В работе [13] авторы проделали большую работу и представили систему обнаружения ВПО в виртуальных средах на базе гипервизора, описали как теоретические, так и практические аспекты реализации системы, однако несмотря на распространенность виртуальных сред, конечные пользователи по-прежнему, чаще всего используют физические компьютеры. В работе [14] для ОС Android авторы описали как архитектуру системы обнаружения ВПО, так и подробности ее реализации, аналогичная ситуация имеет место быть в работе [15], за исключением того, что метод на основе которого разработана система не предполагает поведенческого анализа.

Обобщая все сказанное, можно сказать, что в целом вопрос создания автоматизированных систем обнаружения ВПО раскрыт слабо, особенно слабо по отношению к методам, предполагающим поведенческий анализ, что является большим упущением. Либо такие системы реализуются на основе более высокоуровневой ОС Android, или предполагают совершенно другую среду своей работы, например, как часть гипервизора. Таким образом, видится пробел в области создания и описания систем автоматизированного обнаружения ВПО, в основе которых лежит поведенческий анализ.

В данной работе представлено описание автоматизированной системы обнаружения ВПО, начиная с того метода, который лежит в ее основе, далее – требований к тому, какие данные требуются для работы метода и как реализовать сбор и подготовку данных. Так же раскрываются конкретные технические решения, которые требуются для реализации этой системы. Описываемая автоматизированная система обнаружения ВПО реализует предлагаемый авторами метод обнаружения и кластеризации ВПО, была применена для проведения экспериментальных исследований предложенного метода. Система позволяет осуществлять оперативное обнаружение неизвестных ранее угроз и вместе с тем, за счет проведения кластеризации, идентифицировать конкретную угрозу для реализации наиболее подходящих мер защиты от этой угрозы. Перед тем, как приступить к рассмотрению системы обнаружения ВПО в работе рассмотрен метод, лежащий в ее основе, это позволит получить необходимый контекст, для понимания того, что послужило основанием для требований к разработанной системе обнаружения ВПО.

Метод обнаружения и кластеризации вредоносного программного обеспечения. В основе автоматизированной системы обнаружения ВПО лежит метод обнаружения и кластеризации ВПО, позволяющий выполнять эффективное обнаружение и кластеризацию вредоносного программного обеспечения с использованием признаков декларируемого и фактического функционала. Основой метода является использование декларируемого и фактического функционала для сопоставления, структурирования и устранения шума, в результате чего появляется возможность нахождения признаков, которые позволяют идентифицировать и эффективно выделять семейства ВПО. Образ декларируемого и фактического функционала представляет собой интегрированные метаданные в традиционные трассы системных вызовов (WINAPI в режиме пользователя). Такими метаданными является информация о способе линковки, модуле-источнике вызова, информация о процессе и потоке, в которых произведен вызов системной функции.

Предлагаемый метод состоит из следующих этапов:

- Сбор данных о декларируемом функционале исследуемого образца путем проведения статического анализа, в том числе соответствующих метаданных системных функций: информация о способе линковки.
- Сбор данных о фактическом функционале образца путем проведения поведенческого анализа, в том числе соответствующих метаданных системных функций: трассы вызванных системных функций, информации о способе линковки, информации о модуле-источнике вызова, информации о процессе и потоке, в которых произведен вызов.
- Обнаружение ВПО, основанное на сопоставлении декларируемого и фактического функционала;
- Кластеризация найденных на 3-м этапе экземпляров ВПО, основанная на анализе фактического функционала образца.

Графическое представление данных этапов приведено на (рис. 1).

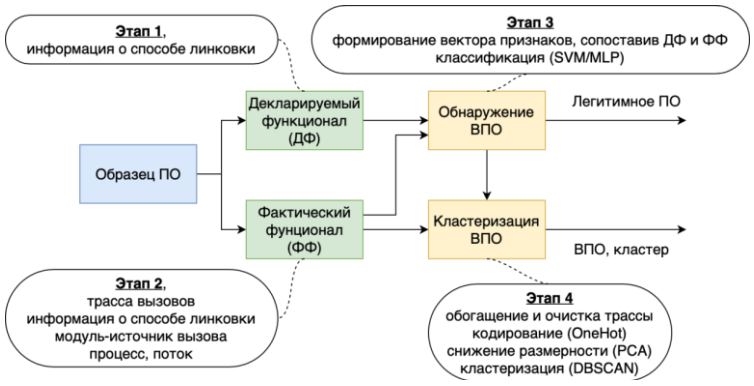

Рис. 1. Этапы работы метода обнаружения и кластеризации ВПО

Предлагаемый метод обнаружения и кластеризации ВПО в части обнаружения ВПО основывается на данных статического и поведенческого анализа. За счет выявления разницы декларируемого функционала (получен в результате статического анализа) и фактического (получен в результате поведенческого анализа) удается идентифицировать ВПО.

На (рис. 2) показана схема обработки потоков данных при анализе образцов ПО в рамках предлагаемого метода обнаружения ВПО.

Рис. 2. Схема обработки потоков данных при анализе образцов ПО в рамках предлагаемого метода обнаружения ВПО

Как говорилось ранее, исходными данными для метода являются результаты статического и поведенческого анализа. Как результат статического анализа – получение списка тех системных функций, которые линкуются в процессе загрузки программы, то есть функции из таблицы импорта исполнимого файла. В результате поведенческого анализа извлекается следующая информация:

- данные о списке функций, связывание которых происходило в процессе исполнения программы путем вызова функции GetProcAddress;
- данные о списке функций, связывание которых происходило в обход описанных ранее стандартных механизмов линковки на этапе загрузки программы и линковки путем вызова функции GetProcAddress. В данном случае линковка производится фактически вручную, путем реализации кода, схожего с кодом функции GetProcAddress;
- данные о модуле – источнике вызова. Каждая системная функция вызывается из участка адресного пространства, этот участок должен принадлежать конкретному исполнимому модулю, в том числе анализируемому, или используемой библиотеке.

Сопоставление полученных данных, позволяет идентифицировать образцы ВПО. Сопоставление реализуется через набор признаков в векторе $P_i = \{p_{1_i}, p_{2_i}, p_{3_i}, p_{4_i}\}$, который характеризует каждый исследуемый образец. Результатом обработки данного вектора является отнесение образца, которому принадлежит соответствующий вектор, к ВПО или легитимному ПО.

Следующим составляющим метода обнаружения и кластеризации ВПО является алгоритм кластеризации ВПО. За счет глубокой интеграции метаданных вызываемых системных функций алгоритм позволяет достичь лучшего качества кластеризации ВПО в сравнении с другими. Интегрируя метаданные в традиционную трассу вызовов системных функций, трасса структурируется, осуществляется ее очистка от зашумляющих данных. Таким образом, предоставляя для кластеризации максимально статичные для каждого семейства ВПО векторы признаков.

Для предлагаемого алгоритма в качестве исходных данных используется трасса выполненных системных вызовов, полученная в результате поведенческого анализа, а также метаданные о каждом вызове.

Под метаданными о вызываемых системных функциях понимается:

- информация о процессе и потоке, в которых выполняется вызов системной функции;
- информация об исполнимом модуле, в котором выполняется вызов системной функции.

На (рис. 3) показана структура предлагаемого алгоритма кластеризации ВПО.

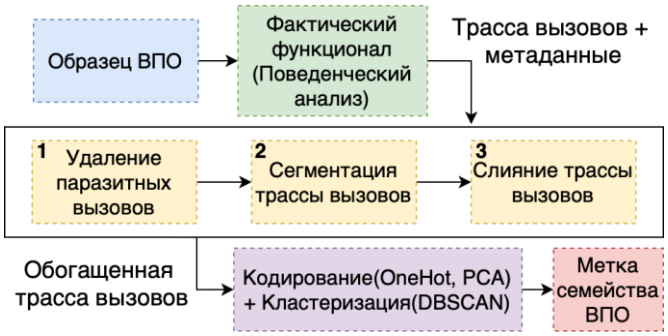

Рис. 3. Общая структура алгоритма кластеризации ВПО

Ядро алгоритма представлено в средней секции, где выполняются все структурные преобразования трассы вызовов:

- Отражаются данные об исполнимом модуле, в котором выполнен вызов системной функции. Удаляя паразитные вызовы, не принадлежащие исполнимому модулю исследуемого образца, устраняются «зашумляющие» данные из трассы вызовов, и функциональные части образца выделяются более четко.
- Выполняется предварительное формирование трассы для каждого потока и процесса отдельно, то есть сегментация трассы вызовов, тем самым добавляя в конечную трассу данные о процессе и потоке, в которых выполнен вызов системной функции.
- Выполняется последовательное слияние трасс вызовов каждого потока. Слияние фактически дополняет предварительную сегментацию, т.к. очень важно сформировать именно тот порядок вызовов, который будет фиксированным для данного образца. Только тогда сопоставление трасс будет корректным. Ограничивающие условия, для выполнения выравнивания, получены экспериментально, на основе статистического анализа результатов поведенческого анализа ВПО.

Далее, обогащённая трасса вызовов системных функций кодируется и выполняется кластеризация, результатом которой является метка семейства ВПО.

Структура автоматизированной системы обнаружения ВПО. Общую схему работы системы и ее архитектуру можно представить так, как показано на (рис. 4).

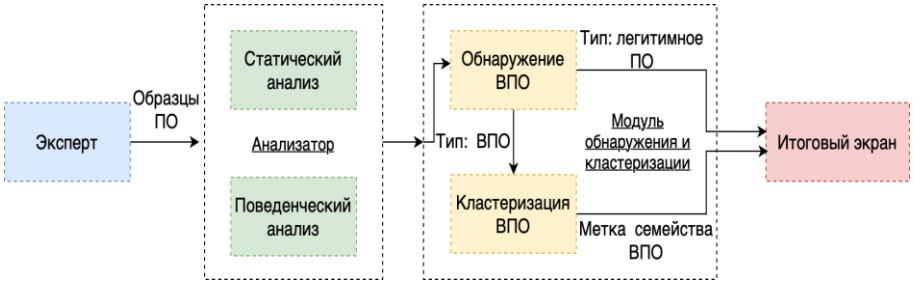

Рис. 4. Структура системы обнаружения ВПО

Компонент «Модуль обнаружения и кластеризации» реализует метод обнаружения ВПО[2] и алгоритм кластеризации ВПО [3], которые кратко описаны ранее. Далее будет рассмотрена реализация компонента «Анализатор», функционал этого компонента обусловлен требованиям к реализации для метода обнаружения и кластеризации ВПО.

Для реализации метода обнаружения и кластеризации ВПО требуется получение следующих сведений об анализируемом образце:

- данные о способе линковки;
- информация о вызываемых системных функциях;
- информация об исполнимом модуле, в котором вызвана рассматриваемая функция;
- информация о контексте вызова, а именно о процессе и потоке в которых вызвана контролируемая функция.

Для получения всех обозначенных данных требуется проведение статического и поведенческого анализа. В рамках поставленной задачи, проведение статического анализа не является сложной задачей, существует множество библиотек на разных языках программирования для извлечения данных, которые требуются для работы предложенного метода. Поведенческий анализ более сложная задача, создание качественных средств поведенческого анализа является самостоятельной и достаточно трудоемкой задачей [16, 17]. Поэтому, первостепенной задачей ставилось подобрать готовую систему поведенческого анализа, которая смогла бы удовлетворить всем требованиям.

Подбор системы поведенческого анализа. Системы поведенческого анализа, так же именуются как песочницы (sandbox), на данных момент достаточно функциональны и обладают дружественным интерфейсом. Однако, наиболее интересны они в разрезе тех характеристик, которые необходимы для реализации предложенной в работе автоматизированной системы обнаружения ВПО. В (табл. 1) приведены сравнительные характеристики трех популярных на данный момент песочниц в контексте необходимых для работы предложенного метода обнаружения и кластеризации данных и других важных критериев.

Таблица 1

Сравнительная характеристика систем поведенческого анализа

	CrowdStrike Falcon Sandbox	**Joebox**	**Cuckoo Sandbox**
Информация о вызываемых системных функциях	+	+	+
Информация о способе линковки	+	+	+
Информация о модуле-источнике вызова	-	-	-
Информация о контексте вызова	-	+/-(для некоторых вызовов идентификатор потока не предоставляется)	-
Исходный код	закрытый	закрытый	открытый
Уровень контроля вызываемых системный функций	системный	системный	пользовательский

На первый взгляд, достаточно много в представленных системах не реализовано, однако, открытость исходного кода Cuckoo Sandbox позволяет доработать ее до нужной функциональности. Прочие представленные системы имеют очень широкий функционал и дают много информации о работе исследуемого образца, однако, такое четкое ориентирование, с другой стороны, лишает исследователей полезной информации, превращая инструмент исследователя в инструмент прикладной работы специалиста по безопасности.

Выбрав Cuckoo Sandbox, рассмотрим ее архитектуру, достоинства и недостатки подробнее. Cuckoo Sandbox представляет собой клиент-серверное решение, где управляющий сервер отвечает одновременно за ход поведенческого анализа в виртуальной среде, управление виртуальной средой, и за сбор данных и создание отчетов об анализе. Серверная часть реализована на языке Python. Клиентская часть представляет собой набор программ-агентов для разных виртуальных сред, которые непосредственно осуществляют мониторинг за работой исследуемых образцов. Для ОС Windows клиентская часть реализована на языке C под MiniGW. То есть исходный код не содержит никаких закрытых участков и лицензий на использование. Агент под ОС Windows функционирует в пользовательском режиме, а не в системном режиме (режиме ядра) как это сделано у конкурентных решений. Это не является проблемой, в тоже время это позволяет использовать официальное решение Microsoft для выполнения мониторинга за вызываемым системными функциями – метод сплайсинга [18], который заключается в модификации первых 5 байт вызываемой функции (пролог) в памяти процесса, так что перед выполнением кода функции выполняется переход на функцию-перехватчик, регистрирующую данный вызов. Хочется отметить, что работа в режиме пользователя не может быть расценена как компромиссное решение, это полноценное решение. Агент, функционирующий в режиме ядра, обладает как плюсами, в виде скрытного (так

же спорный и дискуссионный момент), так и минусами, в виде невозможности корректно и официальными методами отследить трассировку стека вызовов анализируемого образца, это очень сильно сокращает объем той информации, который может быть получен в результате поведенческого анализа. Так же, исходный код агента содержит список прототипов перехватываемых системных функций, который может быть дополнен. Cuckoo Sandbox в целом, законченное решение для поведенческого анализа ВПО, однако не всегда стабильно работающее при использовании множества виртуальных сред, и ограничено по функционалу в части собираемых данных. Так же, в сравнении с конкурентами, не обладает такой большой базой сигнатур потенциально опасных действий, что затрудняет ее полноценное использование именно специалистами по безопасности, а не исследователями. Однако, как было сказано выше, открытый исходный код делает Cuckoo Sandbox абсолютно идеальной для любого исследователя.

Создание автоматизированной системы обнаружения вредоносного программного обеспечения на основе системы поведенческого анализа. Как было сказано ранее, взяв за основу Cuckoo Sandbox возможно доработать ее таким образом, чтобы появилась возможность реализовать предложенный метод обнаружения и кластеризации ВПО. Такими доработками являются:

- расширение списка контролируемых функций;
- получение информации об исполнимом модуле, в которым выполнен вызов контролируемой функции;
- получение информации о контексте вызова системной функции.

Расширение списка контролируемых функций. В связи широким набором системных функций(например в работе [18] проанализировано более 22 тыс функций WINAPI) и а так же различными типами и подтипами[19], очевидно, что недостаточный охват контролируемых системных функций несет в себе опасность пропуска необходимых данных, слишком большой охват чреват как усложнением обработки, в связи с ростом размерности данных, так и различными «шумами» в данных. При этом немаловажно, чтобы набор регистрируемых функций включал в себя достаточное количество тех, которые используются в ВПО, и тех, которые используются в легитимном ПО. Недостаточный охват системных функций, регистрация вызовов которых может быть произведена, впоследствии может привести к некорректным результатам обнаружения ВПО. При анализе первичных данных в ходе решения задачи обнаружения ВПО выяснилось, что имеет место большое количество ошибок определения легитимных образцов, в то же время ошибка определения вредоносных образцов была существенно ниже. Данный факт подтверждает важность выбора набора контролируемых функций, о чем было сказано выше. В результате анализа образцов, которые были ошибочно обозначены как вредоносные, было выявлено, что в большинстве своем это приложения с графическим интерфейсом, которые после запуска ожидали действий пользователя для продолжения своей работы, соответственно, собранная трасса системных вызовов была достаточно коротка. При дальнейшем анализе исходного кода песочницы Cuckoo Sandbox стал очевиден упор со стороны данного ПО на вредоносный функционал, т.е. в подавляющем большинстве перехватываемых средой функций предполагали взаимодействие с файловой системой, реестром, сетью, перехват функций, ответственных за взаимодействие с пользователем через графический или консольный интерфейс, отсутствовал. Далее с помощью утилиты API Monitor был проанализирован каждый неверно классифицированный образец, и выделен набор функций, характерных для приложений с интерфейсом пользователя. Таким образом, список перехватываемых функций был расширен с 336 до 438.

Получение информации об исполнимом модуле. Получение информации об исполнимом модуле, в котором выполнен вызов – задача исключительно техническая. Для этого необходимо извлечь адрес возврата (то есть то место, откуда была вызвана контролируемая функция) из стека, после чего на основе этого адреса получить имя исполнимого модуля, адресному пространству которого принадлежит данный адрес. Для получения имени модуля по адресу внутри него также можно воспользоваться различными способами, самым очевидным из них является использование функций GetModuleHandleEx и GetModuleFileName. Однако, данные функции входят в список контролируемых, возможна ситуация рекурсивного вызова. Для того, чтобы принципиально решить данную проблему, получение данных о модуле, в котором произведен вызов выполняется вручную, путем разбора структуры PEB и сопоставления адресов начала и конца адресного пространства загруженных модулей, в случае если адрес – источник контролируемого вызова находится в пределах начала и конца адресного пространства одного из модулей, источником вызова считается именно этот модуль.

Получение информации о контексте вызова системной функции. Под контекстом подразумевается процесс и поток, в котором выполнен вызов. Получение информации и процессе уже реализована в системе Cuckoo Sandbox. Информация же об потоке – может быть получена путем вызова функций WINAPI GetCurrentThreadId.

После того, как технические проблемы были решены для Cuckoo Sandbox разработано расширение, помогающее оператору производить анализ образцов ВПО с использованием предложенного метода обнаружения и кластеризации ВПО. В частности, основные дополнения:

- класс Classifier (файл cuckoo/modules/reporting/classifierparser.py) класс для свертки сырых данных о результатах анализа из БД MongoDB, обработанные данные помещаются в соответствующие поле результатов анализа;
- класс SampleInfo (файл cuckoo/lib/cuckoo/core/database.py) модель данных SQLAlchemy, представляющая образец, анализ которого производится предложенным авторами метода;
- расширен белый список перехваченных функций, которые учитывает серверная часть (файл cuckoo/lib/cuckoo/common/logtbl.py);
- класс SoftwareClassifier (файл cuckoo/web/classifier/utils.py) класс для выявления образцов ВПО;
- класс MalwareClassifier (файл cuckoo/web/classifier/utils.py) класс для кластеризации ВПО.

А также модифицирован код агента, выполняющего мониторинг запущенного процесса в гостевой системе, в частности:

- реализована функция HMODULE get_module_handle_from_address(PVOID addr) (файл monitor-master/src/misc.c) для получения дескриптора загруженного модуля по адресу внутри него, что в свою очередь нужно для определения модуля в котором выполнен вызов перехваченной функции;
- реализована функция const wchar_t *get_module_file_name_from_address(void *addr) (файл monitor-master/src/misc.c) для получения имени загруженного модуля по дескриптору;
- дополнена функция void log_api(uint32_t index, int is_success, uintptr_t return_value, uint64_t hash, last_error_t *lasterr, ...) (файл monitor-master/src/log.c) в части формирования записи о перехваченной функции добавлена информация о потоке, в котором выполнен вызов и информации о модуле, в котором выполнен вызов перехваченной функции.

Для анализа собранных данных использовался пакет scikit-learn [20] на языке Python, в котором имеются реализации всех используемых математических алгоритмов. Расширение для Cuckoo Sandbox разработано на языке Python, веб-интерфейс реализован с использованием фреймворка Django. Установка системы для анализа производилась на сервер под управлением ОС Ubuntu Server 12. В качестве гостевой ОС, в которой выполнялся запуск исследуемых образцов, использована Windows 7 x64. На (рис. 5) приведен экран разработанного расширения.

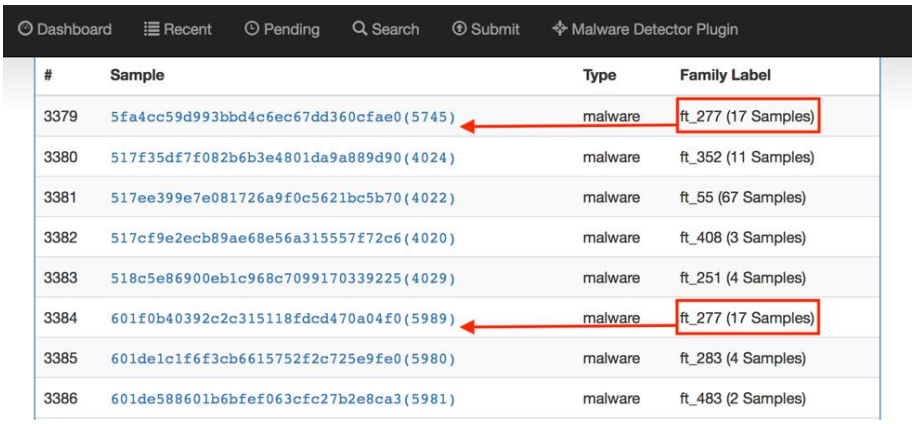

Рис. 5. Экран разработанного расширения, реализующего обнаружение и кластеризацию ВПО

На данном экране представлены данные о последних проанализированных образцах. В качестве идентификатора образца в поле «Sample» используется его md5-сумма, а также номер выполненной задачи по анализу данного образца, указанный в скобках. Для каждого образца в поле «Type» указан его тип:

- Malware – вредоносных образец;
- Benign – легитимный образец.

Поле «Label» содержит назначенную метку класса вредоносного ПО.

Обратившись к данному экрану, эксперт может оперативно понять вердикт системы и при надобности, кликнув на нужный образец в поле «Sample» перейти на экран с подробным отчетом о проанализированном экземпляре.

Возможность портирования системы анализа и обнаружения ВПО. Как было сказано ранее, предложенный авторами метод ориентирован на ОС Windows платформы Win32/Win64. Поэтому возникает логичный вопрос о применимости метода и технический решений, рассмотренных в данной работе на другие ОС или платформы. Выделим несколько основных типов сред для выполнения программ(платформ), которые могут быть тесно связаны с ОС:

1) среда компилируемых в машинный код программ (Win32, Win64, Linux32, Linux64);
2) среда на основе виртуальных машин (.NET, Java(Android[21])).

Из этого разделения и обозначенных ранее условий работы предложенного авторами метода видно, что предложенный метод и большинство технологических решений могут быть применены для любой среды, где выполняются компилируемые программы. Среды под управлением виртуальных машин, такие как .NET, Java, Android, имеют достаточно схожую архитектуру. На (рис. 6) показана схема отношения программ, разработанных в среде .NET в обычным Win32/Win64 программам.

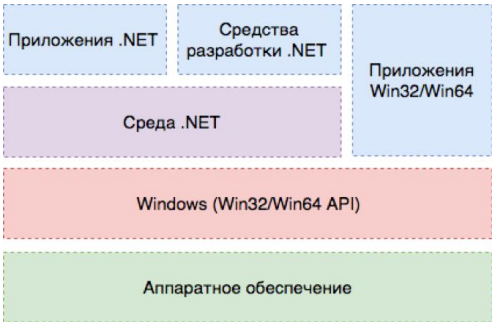

Рис. 6. Схема отношения программ, разработанных в среде .NET в обычным Win32/Win64 программам

Отсюда видно, что отличие сред на основе виртуальных машин заключается в дополнительном уровне программного интерфейса. Таким образом, для предложенного метода в части кластеризации ВПО существует возможность выполнить адаптацию под данные среды за счет перехода от перехвата функций WINAPI на интерфейсные функции соответствующей платформы. Для метода в части обнаружения ВПО возможность провести адаптацию также существует. Рассмотрим подробнее на примере .NET. Для Java, Android ситуация принципиально аналогичная.

Несмотря на то, что линковки в том, виде что есть для Win32/Win64 в .NET не существует, в .NET реализован механизм рефлексии, который позволяет использовать другие сборки(внешние библиотеки) не сообщая об этом заранее, строка с именем загружаемой сборки может быть зашифрована и не доступна для статического анализа, аналогично механизму LoadLibrary/GetProcAddress для Win32/Win64. Так же .NET существует механизм p/invoke который позволяет выполнять вызовы функций из сторонних и в том числе системных библиотек за пределами среды .NET, этот механизм работает в процессе запуска программы, то есть, как только встретится соответствующая инструкция, будет произведен соответствующий вызов, это полностью эквивалентно тому, что программа для свой работы с внешними библиотеками использует связку LoadLibrary/GetProcAddress. Таким образом, для сред на основе виртуальных машин существуют прямые аналоги механизмов, работа с которыми лежит в основе метода обнаружения ВПО. Следовательно, предложенный авторами метод так же может быть адаптирован для сред на основе виртуальных машин.

Касательно технической части, а именно портирования предложенной системы. Для этого, требуется разработать дополнительные модули-агенты в ОС, которые позволили бы выполнить перехват нужных функций и сбор данных, при этом, серверная часть остается не изменой.

Заключение. В данной работе была описана реализация автоматизированной системы обнаружения ВПО, представлена ее структура, раскрыты технические требования к проектируемой системе, обусловленные предложенными ранее методом обнаружения, выполнен подбор необходимых средств реализации, описана их доработка. Так же, раскрыт вопрос возможности портирования данной системы для работы с образцами ВПО разработанными под различные платформы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Babenko L., Kirillov A.* Malware detection by meta-information of used system functions // In Proceedings of the 10th International Conference on Security of Information and Networks. – ACM. 2017. – P. 240-244. – DOI: 10.1145/3136825.3136897.
2. *Babenko L, Kirillov A.* Development of method for malware classification based on statistical methods and an extended set of system calls data // In Proceedings of the 11th International Conference on Se- curity of Information and Networks. – ACM. 2018. – Art. no 8. – DOI: 10.1145/3264437.3264478.
3. Патент № 2535175 C2 Российская Федерация, МПК G06F 21/56. Система и способ обнаружения вредоносного программного обеспечения путем создания изолированной среды: № 2012156433/08: заявл. 25.12.2012: опубл. 10.12.2014 / В.В. Яблоков, Е.Ю. Елисеев ; заявитель Закрытое акционерное общество "Лаборатория Касперского".
4. Патент № 2430411 C1 Российская Федерация, МПК G06F 21/00, G06F 12/00. Система и способ обнаружения вредоносного программного обеспечения: № 2010107437/08: заявл. 02.03.2010: опубл. 27.09.2011 / О.В. Зайцев; заявитель Закрытое акционерное общество "Лаборатория Касперского".
5. *Переберина А.А., Костюшко А.В.* Проектирование программно-аппаратного комплекса для запуска вредоносного программного обеспечения // Тр. Московского физико-технического института (национального исследовательского университета). – 2018. – Т. 10. – № 2 (38). – С. 114-130.
6. *Lin C.H., Pao H.K., Liao J.W.* Efficient dynamic malware analysis using virtual time control mechanics // Computers & Security. – 2018. – Vol. 73. – P. 359-373. – DOI: 10.1016/j.cose.2017.11.010.
7. *Токарев В.Л., Сычугов А.А*. Вариант системы оперативного обнаружения malware // Известия Тульского государственного университета. Технические науки. – 2017. – № 10. – С. 186-195.
8. *Mirza Q.K.A., Awan I., Younas M.* CloudIntell: An intelligent malware detection system // Future Generation Computer Systems. – 2018. – Vol. 86. – P. 1042-1053. – DOI: 10.1016/j.future.2017.07.016.
9. *Baptista I., Shiaeles S., Kolokotronis N.* A novel malware detection system based on machine learning and binary visualization // 2019 IEEE International Conference on Communications Workshops (ICC Workshops). – IEEE, 2019. – P. 1-6. – DOI: 10.1109/ICCW.2019.8757060.
10. *Belaoued M. et al.* Malware detection system based on an in-depth analysis of the portable executable headers // International conference on machine learning for networking. – Springer, Cham, 2018. – P. 166-180. – DOI: 10.1007/978-3-030-19945-6_11.
11. *Ali M. et al.* MALGRA: Machine learning and N-gram malware feature extraction and detection system // Electronics. – 2020. – Vol. 9, No. 11. – P. 1777. – DOI: 10.3390/electronics9111777.
12. *Kumara A., Jaidhar C.D.* Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM // Future Generation Computer Systems. – 2018. – Vol. 79. – P. 431-446. – DOI: 10.1016/j.future.2017.06.002.
13. *Feng P. et al.* A novel dynamic Android malware detection system with ensemble learning // IEEE Access. – 2018. – Vol. 6. – P. 30996-31011. – DOI: 10.1109/ACCESS.2018.2844349.
14. *Hou S. et al.* Make Evasion Harder: An Intelligent Android Malware Detection System // IJCAI. – 2018. – P. 5279-5283. – DOI: 10.24963/ijcai.2018/737.
15. *Hunt G., Brubacher D.* Detours: Binary interception of win32 functions. – 3rd usenix windows nt symposium. – 1999.
16. *Or-Meir O. et al.* Dynamic malware analysis in the modern era–A state of the art survey // ACM Computing Surveys (CSUR). – 2019. – Vol. 52, No. 5. – P. 1-48. – DOI: 10.1145/3329786.
17. *Egele M. et al.* A survey on automated dynamic malware-analysis techniques and tools // ACM computing surveys (CSUR). – 2008. – Vol. 44, No. 2. – P. 1-42.
18. *Jiang H., Turki T., Wang J.T.L.* DLGraph: Malware detection using deep learning and graph embedding // 2018 17th IEEE international conference on machine learning and applications (ICMLA). – IEEE, 2018. – P. 1029-1033. – DOI: 10.1109/ICMLA.2018.00168.
19. *Russinovich M.E., Solomon D.A., Ionescu A.* Windows internals. Part 2. – Pearson Education, 2012.
20. *Pedregosa F. et al.* Scikit-learn: Machine learning in Python // Journal of machine Learning research. – 2011. – Vol. 12. – P. 2825-2830. – DOI: 10.1145/2089125.2089126.
21. *Brahler S.* Analysis of the android architecture // Karlsruhe institute for technology. – 2010. – Vol. 7, No. 8.

REFERENCES

1. *Babenko L., Kirillov A.* Malware detection by meta-information of used system functions, *In Proceedings of the 10th International Conference on Security of Information and Networks*, ACM. 2017, pp. 240-244. DOI: 10.1145/3136825.3136897.
2. *Babenko L, Kirillov A.* Development of method for malware classification based on statistical methods and an extended set of system calls data, *In Proceedings of the 11th International Conference on Se- curity of Information and Networks*. ACM. 2018, Art. no 8. DOI: 10.1145/3264437.3264478.
3. *YAblokov V.V., Eliseev E.Yu.* Patent № 2535175 C2 Rossiyskaya Federatsiya, MPK G06F 21/56. Sistema i sposob obnaruzheniya vredonosnogo programmnogo obespecheniya putem sozdaniya izolirovannoy sredy: № 2012156433/08: zayavl. 25.12.2012: opubl. 10.12.2014; zayavitel' Zakrytoe aktsionernoe obshchestvo "Laboratoriya Kasperskogo" [Patent No. 2535175 C2 Russian Federation, MPC G06F 21/56. System and method for detecting malware by creating an isolated environment: No. 2012156433/08 : Appl. 12/25/2012 : publ. December 10, 2014; Applicant Closed Joint Stock Company "Kaspersky Laboratory"].
4. *Zaytsev O.V.* Patent № 2430411 C1 Rossiyskaya Federatsiya, MPK G06F 21/00, G06F 12/00. Sistema i sposob obnaruzheniya vredonosnogo programmnogo obespecheniya: № 2010107437/08: zayavl. 02.03.2010: opubl. 27.09.2011; zayavitel' Zakrytoe aktsionernoe obshchestvo "Laboratoriya Kasperskogo" [Patent No. 2430411 C1 Russian Federation, MPC G06F 21/00, G06F 12/00. Malware detection system and method: No. 2010107437/08: Appl. 03/02/2010: publ. 27.09.2011; Applicant Closed Joint Stock Company "Kaspersky Laboratory"].
5. *Pereberina A.A., Kostyushko A.V.* Proektirovanie programmno-apparatnogo kompleksa dlya zapuska vredonosnogo programmnogo obespecheniya [Designing a hardware-software complex for launching malware], *Tr. Moskovskogo fiziko-tekhnicheskogo instituta (natsional'nogo issledovatel'skogo universiteta)* [Proceedings of the Moscow Institute of Physics and Technology (National Research University)], 2018, Vol. 10, No. 2 (38), pp. 114-130.
6. *Lin C.H., Pao H.K., Liao J.W.* Efficient dynamic malware analysis using virtual time control mechanics, *Computers & Security*, 2018, Vol. 73, pp. 359-373. DOI: 10.1016/j.cose.2017.11.010.
7. *Tokarev V.L., Sychugov A.A.* Variant sistemy operativnogo obnaruzheniya malware [A variant of the malware detection system], *Izvestiya Tul'skogo gosudarstvennogo universiteta. Tekhnicheskie nauki* [Bulletin of the Tula State University. Technical science], 2017, No. 10, pp. 186-195.
8. *Mirza Q.K.A., Awan I., Younas M.* CloudIntell: An intelligent malware detection system, *Future Generation Computer Systems*, 2018, Vol. 86, pp. 1042-1053. DOI: 10.1016/j.future.2017.07.016.
9. *Baptista I., Shiaeles S., Kolokotronis N.* A novel malware detection system based on machine learning and binary visualization, *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2019, pp. 1-6. DOI: 10.1109/ICCW.2019.8757060.
10. *Belaoued M. et al.* Malware detection system based on an in-depth analysis of the portable executable headers, *International conference on machine learning for networking*. Springer, Cham, 2018, pp. 166-180. DOI: 10.1007/978-3-030-19945-6_11.
11. *Ali M. et al.* MALGRA: Machine learning and N-gram malware feature extraction and detection system, *Electronics*, 2020, Vol. 9, No. 11, pp. 1777. DOI: 10.3390/electronics9111777.
12. *Kumara A., Jaidhar C.D.* Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM, *Future Generation Computer Systems*, 2018, Vol. 79, pp. 431-446. DOI: 10.1016/j.future.2017.06.002.
13. *Feng P. et al.* A novel dynamic Android malware detection system with ensemble learning, *IEEE Access*, 2018, Vol. 6, pp. 30996-31011. DOI: 10.1109/ACCESS.2018.2844349.
14. *Hou S. et al.* Make Evasion Harder: An Intelligent Android Malware Detection System, *IJCAI*, 2018, pp. 5279-5283. DOI: 10.24963/ijcai.2018/737.
15. *Hunt G., Brubacher D.* Detours: Binary interception of win32 functions. 3rd usenix windows nt symposium, 1999.
16. *Or-Meir O. et al.* Dynamic malware analysis in the modern era–A state of the art survey, *ACM Computing Surveys (CSUR)*, 2019, Vol. 52, No. 5, pp. 1-48. DOI: 10.1145/3329786.
17. *Egele M. et al.* A survey on automated dynamic malware-analysis techniques and tools, *ACM computing surveys (CSUR)*, 2008, Vol. 44, No. 2, pp. 1-42.

18. *Jiang H., Turki T., Wang J.T.L.* DLGraph: Malware detection using deep learning and graph embedding,/ *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 2018, pp. 1029-1033. DOI: 10.1109/ICMLA.2018.00168.
19. *Russinovich M.E., Solomon D.A., Ionescu A.* Windows internals. Part 2. Pearson Education, 2012.
20. *Pedregosa F. et al.* Scikit-learn: Machine learning in Python, *Journal of machine Learning research,* 2011, Vol. 12, pp. 2825-2830. DOI: 10.1145/2089125.2089126.
21. *Brahler S.* Analysis of the android architecture, *Karlsruhe institute for technology*, 2010, Vol. 7, No. 8.

Статью рекомендовал к опубликованию д.т.н., профессор Я.Е. Ромм.

Бабенко Людмила Климентьевна – Южный федеральный университет, e-mail: lkbabenko@sfedu.ru г. Таганрог, Россия; д.т.н.; профессор.

Кириллов Алексей Сергеевич – e-mail: kirillovalexeys@gmail.com; программист.

Babenko Lyudmila Kliment'evna – Southern Federal University; e-mail: lkbabenko@sfedu.ru; Taganrog, Russia; dr. of eng. sc.; professor.

Kirillov Alexey Sergeevich – e-mail: kirillovalexeys@gmail.com; programmer.

УДК 004.032.26 DOI 10.18522/2311-3103-2021-7-167-177

С.М. Гушанский, В.Е. Буглов

КВАНТОВОЕ ГЛУБОКОЕ ОБУЧЕНИЕ СВЁРТОЧНОЙ НЕЙРОННОЙ СЕТИ С ИСПОЛЬЗОВАНИЕМ ВАРИАЦИОННОЙ КВАНТОВОЙ СХЕМЫ

Квантовый компьютинг в общем и квантовое глубокое обучение, в частности, представляют собой перспективную область, связанную с исследованиями современных методов и алгоритмов квантовых вычислений, применяемых с целью обучения и разработки новых архитектур искусственных нейронных сетей. В последнее время наблюдается тенденция, состоящая в том, что исследования, проводимые в области квантового глубокого обучения, получают всё большее распространение среди специалистов. Это можно объяснить тем, что было установлено – квантовые схемы способны функционировать подобно искусственным нейронным сетям, демонстрируя при этом лучшие результаты при решении ряда задач, среди которых, например, актуальная задача классификации объектов на изображении или в видеопотоке. Благодаря стремительному развитию квантовых вычислений в области глубокого обучения были найдены оптимальные способы решений для таких актуальных задач, как – проблема исчезающего градиента, нахождение локального минимума, повышение эффективности функционирования крупномасштабных параметрических алгоритмов машинного обучения, устранение декогеренции и квантовых ошибок и пр. В рамках данной работы описан процесс функционирования квантовой вариационной схемы, установлены её основные характеристики и выявлены недостатки. Также проанализированы ключевые особенности квантовых вычислений, на которых основывается процесс реализации квантового глубокого обучения с подкреплением свёрточной нейросети. Кроме того, осуществлено квантовое глубокое обучение свёрточной нейронной сети с помощью применения вариационной квантовой схемы, что приводит к повышению производительности свёрточной нейросети в решении задачи обработки изображения, а именно его классификации, за счёт использования квантовой среды вычислений. Актуальность данной статьи состоит в реализации алгоритма квантового глубокого обучения с подкреплением свёрточной нейросети для обработки изображений, а также большом значении тематики данного исследования для будущей разработки квантовых вычислительных устройств, которые могут быть использованы в системах искусственного интеллекта и т.п., что соответствует приоритетному направлению развития отечественной науки.

Квантовые вычисления; глубокое обучение; свёрточная нейронная сеть; вариационная квантовая схема; исчезающий градиент; алгоритм.