МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

УДК 004.056.57

ОБНАРУЖЕНИЕ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ НА ОСНОВЕ АДАПТИВНО-РЕЗОНАНСНОЙ ТЕОРИИ[1]

Д. Г. Буханов, В. М. Поляков, М. А. Редькина

Белгородский государственный технологический университет им. В. Г. Шухова, г. Белгород, Россия

Рассматривается процесс выявления вредоносного программного кода антивирусными системами. Для анализа исполняемого кода используется граф потока управления. Предлагается в качестве классификатора применять искусственные нейронные сети на основе адаптивно-резонансной теории с иерархической структурой памяти. Для удобного представления графа потока управления при классификации используется алгоритм `graph2vec`. Проведены эксперименты на модельных примерах, которые показали хорошие результаты точности и скорости определения типа вредоносного программного обеспечения.

Ключевые слова: *вредоносное программное обеспечение, анализ исполняемых файлов, граф потока управления, векторизация, деобфускация, искусственная нейронная сеть на базе адаптивной резонансной теории, кластеризация.*

DOI 10.17223/20710410/52/4

DETECTION OF MALWARE USING AN ARTIFICIAL NEURAL NETWORK BASED ON ADAPTIVE RESONANT THEORY

D. G. Bukhanov, V. M. Polyakov, M. A. Redkina

Belgorod State Technological University named after V. G. Shukhov, Belgorod, Russia

E-mail: db.old.stray@gmail.com, p_v_m@mail.ru, redckina.rit@mail.ru

The process of detecting malicious code by anti-virus systems is considered. The main part of this process is the procedure for analyzing a file or process. Artificial neural networks based on the adaptive-resonance theory are proposed to use as a method of analysis. The `graph2vec` vectorization algorithm is used to represent the analyzed program codes in numerical format. Despite the fact that the use of this vectorization method ignores the semantic relationships between the sequence of executable commands, it allows to reduce the analysis time without significant loss of accuracy. The use of an artificial neural network ART-2m with a hierarchical memory structure made it possible to reduce the classification time for a malicious file. Reducing the

[1]Исследование выполнено при финансовой поддержке Минобрнауки России (грант ИБ), проект № 13, и гранта РФФИ № 19-29-09056мк.

classification time allows to set more memory levels and increase the similarity parameter, which leads to an improved classification quality. Experiments show that with this approach to detecting malicious software, similar files can be recognized by both size and behavior.

Keywords: *malware, analysis of portable executable files, control flow graph, vectorization, deobfuscation, artificial neural networks based on adaptive resonance theory, clustering.*

Введение

Вредоносное программное обеспечение (ВПО), проникающее в систему пользователя, может задействовать ресурсы атакуемой ЭВМ в своих целях или привести к потере данных и, следовательно, убыткам пользователей и компаний. В [1] представлены результаты исследований убытков от ВПО. От вируса-вымогателя WannaCry компании понесли потери в 1 млрд долл., а среднегодовые убытки от киберугроз обходятся банкам в 18,28 млн долл. С каждым годом стоимость последствий проведения кибератак растёт, к 2018 г. [2] средняя стоимость инцидента для крупной компании выросла на 24 % (с 992 тыс. долл. до 1,23 млн долл.), а для малого и среднего бизнеса — на 38 % (с 88 до 120 тыс. долл.) по сравнению с предыдущим годом. С ростом угроз растёт сложность определения ВПО. Все чаще при распространении ВПО злоумышленники прибегают к методам сокрытия его присутствия либо к методам социальной инженерии [3].

Основным инструментом при выявлении ВПО является специализированное программное обеспечение — антивирус. Ядром любого антивируса является модуль анализа файлов и процессов. Они базируются на сигнатурном и эвристическом методах выявления зловредного программного обеспечения.

Недостатком сигнатурного анализа [4] является невозможность выявления нового и замаскированного программного обеспечения. Разработчики ВПО прибегают к следующим способам маскировки исполняемых кодов:

1) шифрование [5];
2) самомодификация [6];
3) упаковка [7, 8];
4) обфускация кода [9, 10].

Обфускация представляет собой процесс добавления инструкций, не изменяющих функциональность программы, но изменяющих её размеры и увеличивающих сложность понимания анализируемых инструкций. Далее рассмотрен процесс деобфускации с целью упростить процесс обработки исходного кода.

В отличие от сигнатурного анализа, эвристический используется при выявлении неизвестного ранее ВПО. Основным принципом является нахождение отклонения поведения программы от нормального состояния. Перспективным подходом в построении систем определения ВПО является использование в них искусственных нейронных сетей (ИНС) [11].

ИНС принимает на вход числовые векторы данных, для получения которых исполняемый код ВПО можно представить в виде графа потока управления (ГПУ) [12]. Такая структуризация исполняемого кода позволяет сравнивать вершины друг с другом и выявлять среди них уникальные. Для сравнения графовых структур применяется алгоритм graph2vec [13], позволяющий описать граф вектором чисел.

В работе [14] анализируется накопленная во время выполнения ВПО информация, включающая сетевой трафик, инструкции центрального процессора, дампы оперативной памяти. Авторы используют ИНС Кохонена, основным недостатком которой является необходимость знать заранее число кластеров.

В [15] рассматривается последовательность вызова системных API-функций у 500 исполняемых файлов. При классификации полученных данных несколькими классификаторами лучшую точность обнаружения показал наивный байесовский классификатор. Но он не позволяет выявлять новые виды ранее неизвестного ВПО.

В настоящее время существуют различные архитектуры ИНС, общим недостатком которых является необходимость проводить переобучение, возникающее при добавлении нового образа, недостаточная пластичность и невозможность дообучения в режиме функционирования. Этот недостаток преодолён в [16] при разработке ИНС на основе адаптивной резонансной теории (АРТ).

В работе решается проблема выявления ВПО на основе ИНС адаптивно-резонансной теории с иерархической структурой памяти путём выполнения следующих шагов:

1) деобфускация кода;
2) построение ГПУ;
3) векторизация ГПУ;
4) классификация ВПО по ГПУ.

1. Деобфускация кода

Объектами анализа являются запущенный процесс, дамп его памяти, исполняемый файл или PE-файл (Portable Executable, «переносимый исполняемый»), которые дизассемблируются для изучения и дальнейшей обработки. На первом шаге производится деобфускация кода. Методы анализа при деобфускации программного кода бывают: синтаксические, статические, статистические, динамические [17]. В работе используется метод статического анализа: очистка кода происходит без его выполнения с удалением заранее определенных выражений:

1) команды `nop`, предписывающей ничего не делать;
2) парных команд `<push, pop>`, `<inc, dec>`, `<dec, inc>`, операндами которых являются регистры, не используемые в коде, заключенном между данной парой команд;
3) равнозначные инструкции `xor <operand>,<operand>`, `mov <operand>,0`, если среди инструкций, заключенных между этими двумя, `<operand>` не был упомянут;
4) бесполезные инструкции: `or <operand>,<operand>`; `mov <operand>,<operand>`;
5) команды логического сдвига (`shr`, `shl` и т. д.), `add`, `sub`, второй операнд которых равен нулю;
6) не имеющая смысла пара инструкций `xchng <operand1>,<operand2>` и `xchng <operand2>,<operand1>`, если их операнды не были упомянуты между ними, и др.

Рассмотрим пример деобфускации следующего исполняемого кода:

```
1 401000 sub esp,1C
2 401003 inc edx
3 401004 mov eax,ss:[esp+20]
4 401008 mov eax,ds:[eax]
```

```
 5  40100A mov eax,ds:[eax]
 6  40100C cmp eax,C0000091
 7  401012 ja 401060
 8  401018 cmp eax,C000008D
 9  40101E dec edx
10  40101F shr eax,0
11  401022 jae 401079
12  401028 cmp eax,C0000005
13  40102E add edx,0
14  401031 jne 4010F0
15  401037 nop
16  401038 mov ebx,ebx
17  40103A nop
```

После деобфускации в данном примере значимыми останутся только следующие строки кода:

```
 1  401000 sub esp,1C
 2  401004 mov eax,ss:[esp+20]
 3  401008 mov eax,ds:[eax]
 4  40100A mov eax,ds:[eax]
 5  40100C cmp eax,C0000091
 6  401012 ja 401060
 7  401018 cmp eax,C000008D
 8  401022 jae 401079
 9  401028 cmp eax,C0000005
10  401031 jne 4010F0
```

Были удалены следующие команды: попарные операции `401003 inc edx`, `40101E dec edx`, так как в коде, заключенном между ними, регистра `edx` или его составляющих (`dh, dl`) не найдено; `40101F shr eax,0`; `40102E add edx,0`; `401038 mov ebx,ebx`, которые не изменяют значение первого операнда, следовательно, не имеют смысла; `401037 nop`, `40103A nop` — незначащие операции.

2. Построение ГПУ

На следующем шаге выполняется построение ГПУ. Преобразованный код разбивается на базовые блоки, каждому из которых соответствует вершина ГПУ. Базовые блоки формируются при последовательном анализе кода и не содержат в себе команд управления потоком. Таким образом, вершину можно представить набором команд, из которых состоит базовый блок, соответствующий текущей вершине. Передача управления потоком и конец вершины определяются следующими типами команд:

1) безусловные: `jmp`, `call`, где `jmp` — это команда, приводящая лишь в одну вершину, а `call` работает по принципу условных команд передачи управления;
2) условные: `je`, `jz` и другие, подразумевающие переход в вершину по условию, при его невыполнении переход на следующую команду за данной;
3) команды управления циклом: `loop`, `loope`, `loopz` и другие, работающие по принципу команд условной передачи управления.

Проанализируем участок кода, полученный на предыдущем шаге. Среди набора инструкций можно выделить три вершины, заканчивающиеся условными командами передачи управления: `ja 401060`, `jae 401079`, `jne 4010F0`. ГПУ для заданного участка программы представлен на рис. 1, где показано, что вершин, содержащих адреса

`401060`, `401079`, `4010F0`, в коде не встречено, поэтому символами «???» отмечены неизвестные команды, которые скрыты за соответствующими адресами. Следовательно, переход в эти вершины, расположенные вне рассматриваемого кода, можно отметить как переход в недостижимые: -1.

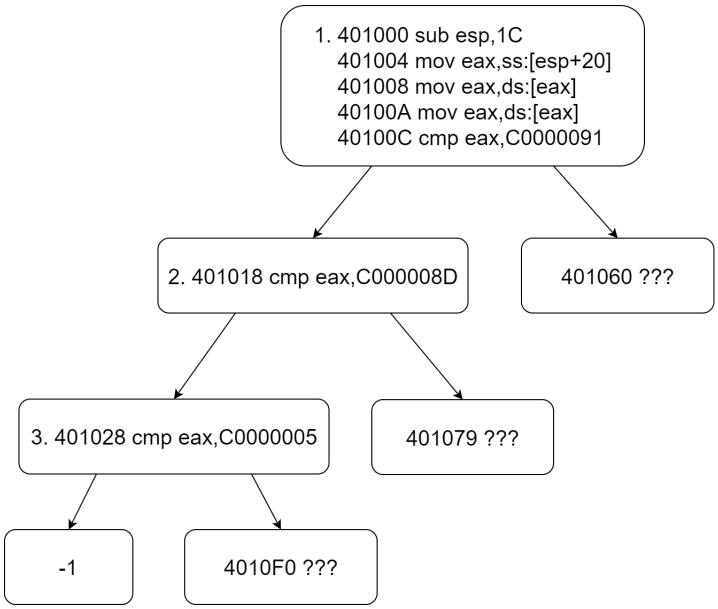

Рис. 1. ГПУ программы

3. Векторизация ГПУ

После выполнения процесса деобфускации кода и составления по нему ГПУ необходимо получить векторное представление графа. Каждую вершину можно описать вектором количеств повторений ассемблерных команд для архитектуры x86-64, из которых она состоит. После сравнения полученных векторов требуется определить уникальные вершины и получить векторное представление ГПУ.

Представим в качестве алфавита команд набор инструкций, используемый в архитектуре x86-64. Для примера алфавит состоит из команд, упомянутых в коде: `sub`, `mov`, `cmp` и др.; W_i — вектор количеств повторений команд, обозначающий i-ю вершину:

$$W_1 = (1, 3, 1), \quad W_2 = W_3 = (0, 0, 1).$$

Уникальных вершин в этом ГПУ $n = 2$. Каждой уникальной вершине присваивается бинарный вектор $VB = (vb_1, \ldots, vb_n)$, одинаковый для повторяющихся вершин; в нём $vb_i = 1$, где i — порядковый номер уникальной вершины, остальные $n - 1$ элементов заполняются нулями. Для нашего примера $VB_1 = (1, 0)$, $VB_2 = VB_3 = (0, 1)$.

Вектор, описывающий данную программу: $V = \sum_{i=1}^{3} VB_i = (1, 2)$. Векторизация ГПУ из примера представлена на рис. 2, где наглядно отображены переходы между вершинами, векторы W_i, V_i, $i = 1, 2, 3$, описывающие соответствующие вершины, а также полученный результирующий вектор V для данной программы.

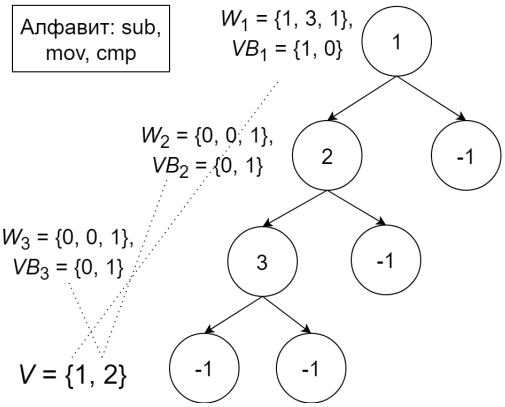

Рис. 2. Векторизация ГПУ

4. Разработка классификатора на основе адаптивно-резонансной теории с иерархической структурой памяти

ИНС на основе АРТ является самоорганизующейся сетью, позволяющей в реальном времени решать задачи кластеризации и распознавания входных образов без учителя. Принцип работы ИНС АРТ основан на нахождении соответствия между восходящим сигналом и ожидаемым нисходящим.

В ИНС АРТ-2 с непрерывными входными сигналами были выявлены недостатки, связанные с низкой скоростью её работы в процессе распознавания образов при большом объёме анализируемых данных. Для решения этой проблемы в работе [18] представлена модификация сети АРТ-2m, имеющая древовидную структуру памяти с рекуррентно изменяющимся параметром сходства для каждого уровня в дереве.

На рис. 3 представлена сеть АРТ-2m, которая состоит из полей F_1, F_2, а также поля сходства G, представленного набором параметров сходства $Riter_i$ для каждого уровня i, где $i = 1, \ldots, max\_level$; max_level — общее количество всех уровней памяти.

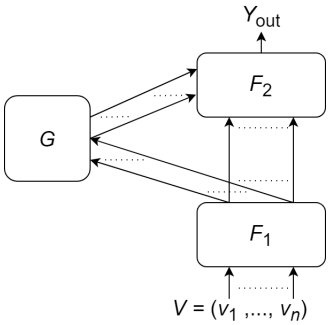

Рис. 3. Схема АРТ-2m

Использование сети с многоуровневой структурой памяти позволяет выполнять поиск активного нейрона быстрее, чем использование классической структуры сети АРТ-2 [16]. Проверка на соответствие происходит только между теми нейронами, которые связаны с нейронами нижележащих уровней. При выявлении ВПО это позволяет уменьшить время идентификации и нахождения зависимостей между уже существующими образами в памяти и поданными на вход.

5. Исследование качества определения ВПО

Проведены эксперименты по моделированию ситуации выявления сходства между ВПО, исходным файлом (ИФ) и ИФ, заражённым этим ВПО. Файлами для экспериментов послужили программа построения ГПУ в качестве ВПО, программа, реализующая принцип работы ART-2m в качестве ИФ, и программа, которая представляет «склейку» ВПО и ИФ, в качестве заражённого ИФ. Данные экспериментов отображены в табл. 1 и 2.

Таблица 1

Исходные данные эксперимента 1

Тип файла	Размер исполняемого кода, кбайт	Общее количество вершин в ГПУ	Количество уникальных вершин
ВПО	202	1596	443
ИФ	160	1256	417
Заражённый файл	363	2851	558

Выходные векторы V_j, описывающие программы, состоят из $n = 558$ элементов и имеют следующий вид:

$$V_1 = (1, 98, 115, 44, 1, \ldots, 0, 0, 0),$$
$$V_2 = (1, 98, 88, 46, 1, \ldots, 1, 1, 0),$$
$$V_3 = (1, 196, 203, 90, 2, 206, \ldots, 1, 0, 1).$$

Начальный параметр сходства $Riter_1 = 0{,}8$. Значение параметров сходства различных уровней определяется следующей рекуррентной зависимостью:

$$Riter_{i+1} = Riter_i + 0{,}7(1 - Riter_i), \quad i = 1, \ldots, max\_level.$$

При $max\_level = 8$ параметры сходства для уровней равны

$$0{,}8,\ 0{,}94,\ 0{,}982,\ 0{,}9946,\ 0{,}99838,\ 0{,}999514,\ 0{,}999854,\ 0{,}999956.$$

Значение параметра сходства $Riter_1 = 0{,}8$ было выбрано эмпирически на основе данных [16]. При значении параметра сходства $\geqslant 0{,}8$ получение новых классов образов происходит с большей вероятностью.

Память представляет набор матриц весовых коэффициентов (z) для каждого уровня иерархии [18]; z-веса для каждого образа вычисляются следующим образом:

1) обучение весов нового нейрона с номером m:

$$z^k_{m\,i} := d \cdot u_i, \quad i = 1, \ldots, n, \quad k = 1, \ldots, max\_level,$$

где u_i — элементы слоя U, входящего в состав поля F_1; k — уровень иерархии;

2) дообучение весов нейрона-победителя с номером $i_{\max}$:

$$z_{i_{\max} i} := z_{i_{\max} i} + d(1 - d)\left(\frac{u_i}{1 - d} - z_{i_{\max} i}\right), \quad i = 1, \ldots, n.$$

В ходе эксперимента для программ получены следующие веса:

1) z-веса ВПО:
   1.1. $z^k_{m\,i}$ = 0,00364339, 0,357052, 0,41899, ..., 0, 0; $k = 1, \ldots, max\_level$, $m = 1$, $i = 1, \ldots, n$;
2) z-веса ИФ:
   2.1. z^k_{mi} = 0,0788018, 0,772257, 0,782974, ..., 0,00456469, 0; $k = 1, \ldots, 4$, $m = 1$, $i = 1, \ldots, n$;
   2.2. z^k_{mi} = 0,00456469, 0,44734, 0,401693, ..., 0,00456469, 0; $k = 5, \ldots$, max_level, $m = 2$, $i = 1, \ldots, n$;
3) z-веса заражённого файла:
   3.1. z^k_{mi} = 0,00925834, 1,11188, 1,13624, ..., 0,004153, 0,00208738; $k = = 1, \ldots, 4$, $m = 1$, $i = 1, \ldots, n$;
   3.2. z^k_{mi} = 0,00540287, 0,734044, 0,805019, ..., 0, 0,00208738; $k = 5$, $m = 1$, $i = 1, \ldots, n$;
   3.3. z^k_{mi} = 0,00208738, 0,409127, 0,423739, ..., 0, 0,00208738; $k = 6, \ldots$, max_level, $m = 3$, $i = 1, \ldots, n$.

На рис. 4 представлена память АРТ-2m сети в виде древовидной структуры. Внутри узлов записаны координаты вершин в декартовой системе. Это позволяет проследить зависимость между объектами исследования: после «заражения» ИФ стал похож на ВПО.

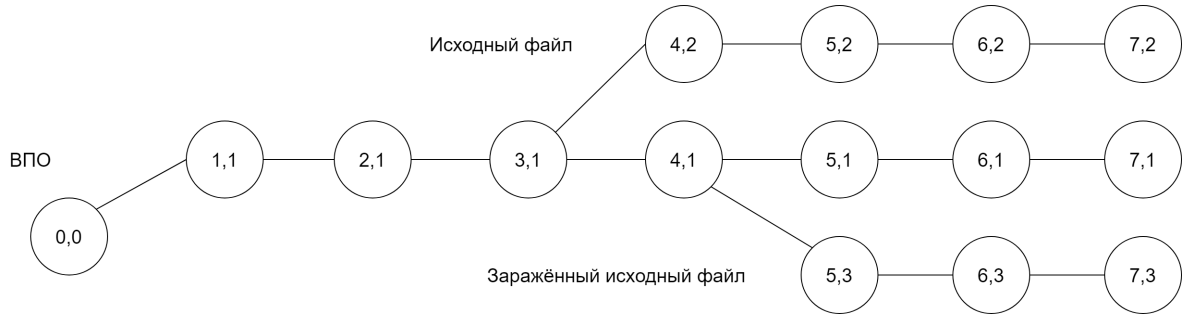

Рис. 4. Структура памяти АРТ-2m (эксперимент 1)

Как видно из рис. 4, заражённый ИФ похож на ВПО с параметром схожести $Riter_5 = 0{,}99838$, а ВПО — на ИФ с $Riter_4 = 0{,}9946$.

В табл. 2 представлены данные для экспериментов, результаты которых изображены на рис. 5–8. Обнаружено, что заражённый файл больше похож на файл, размер которого больше. Полученные графы показывают, что ИФ перестаёт быть похожим на себя. Были использованы четыре типа файлов эквивалентных размеров.

Из рис. 5 видно, что ИФ2 похож на ВПО с параметром схожести $Riter_5 = 0{,}99838$, а Заражённый файл2 — на ИФ2 с $Riter_6 = 0{,}999514$.

Таблица 2

Исходные данные экспериментов 2–5

Эксперимент № п/п	Тип файла	Размер исполняемого кода, кбайт	Общее количество вершин в ГПУ	Количество уникальных вершин	На что больше похож заражённый файл
2	ВПО	202	1596	443	
	ИФ2	204	1693	417	+
	Заражённый файл2	406	3288	506	
3	ВПО	202	1596	443	
	ИФ3	228	1700	427	+
	Заражённый файл3	431	3295	560	
4	ВПО2	204	1693	417	+
	ИФ	160	1256	417	
	Заражённый файл4	365	2948	540	
5	ВПО2	204	1693	417	
	ИФ3	228	1700	427	+
	Заражённый файл5	432	3392	548	

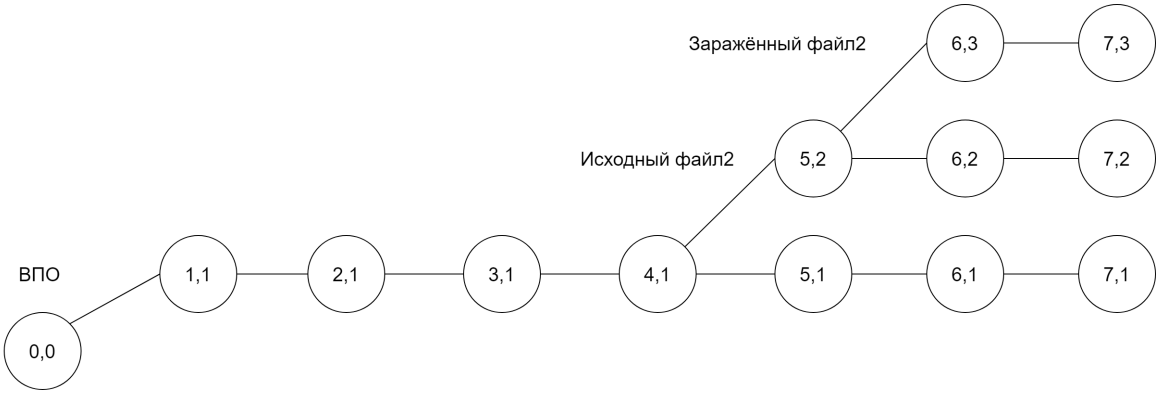

Рис. 5. Результаты эксперимента 2

На рис. 6 показано, что ИФ3 похож на ВПО с параметром схожести $Riter_4 = 0{,}9946$, а Заражённый файл3 — на ИФ3 с $Riter_6 = 0{,}999514$.

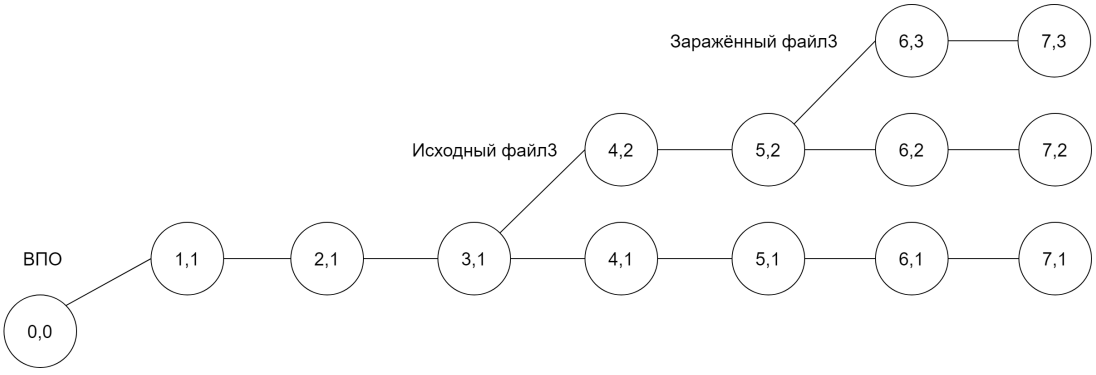

Рис. 6. Результаты эксперимента 3

Из рис. 7 видно, что Заражённый файл4 похож на ВПО2 с параметром схожести $Riter_5 = 0{,}99838$, а ИФ — на ВПО2 с $Riter_4 = 0{,}9946$.

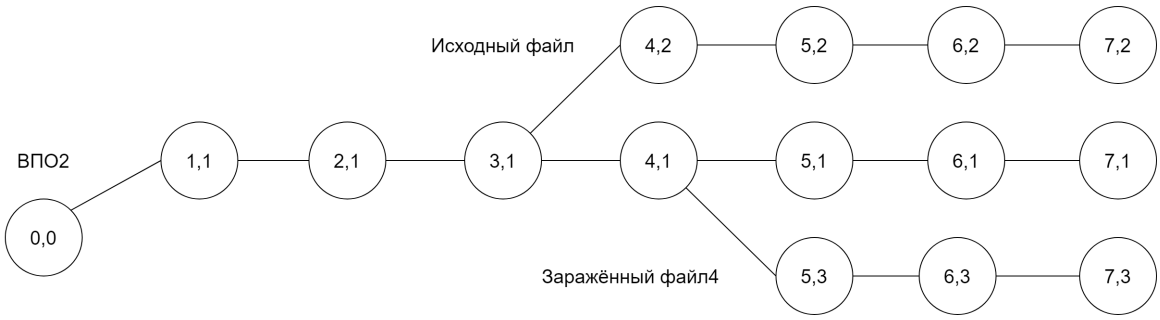

Рис. 7. Результаты эксперимента 4

На рис. 8 показано, что ИФ3 похож на ВПО2 с параметром схожести $Riter_4 = 0{,}9946$, а Зараженный файл5 — на ИФ3 с $Riter_5 = 0{,}99838$.

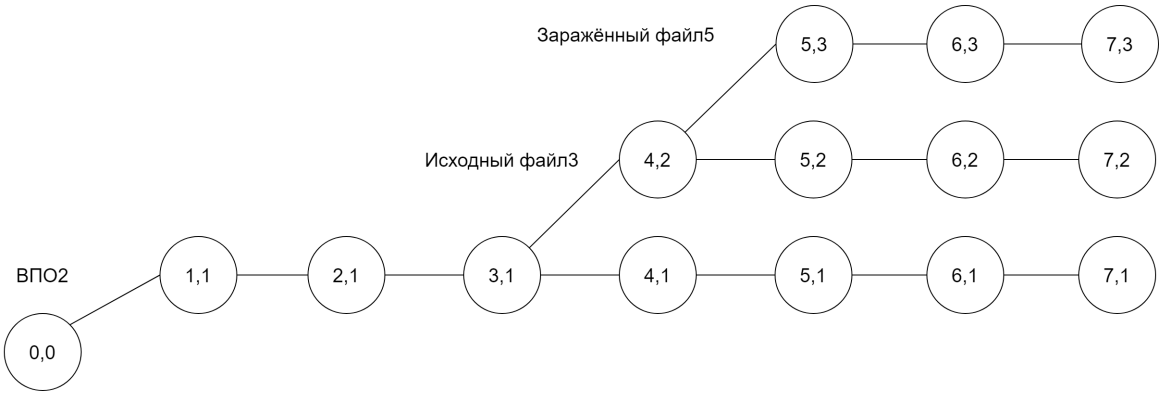

Рис. 8. Результаты эксперимента 5

Результаты экспериментов показывают, что когда размер ИФ превышает размер вредоносного, заражённый файл больше похож на исходный. В этом случае для повышения точности требуется ввести дополнительный уровень памяти. Чем больше уровней, тем выше параметр сходства *Riter* и точность классификации.

Для оценки качества классификации (без учёта фактора размера файла и доли входных данных в конечном файле) проведены дополнительные эксперименты.

6. Исследование качества классификации APT-2m при анализе файлов схожих размеров

В табл. 3 представлены параметры исходных данных эксперимента — приложений и их модификаций (по четыре модификации для каждого приложения).

Первая программа ex_socket_1 выполняет открытие udp socket для всех интерфейсов системы и приём данных. Модификация ex_socket_2, помимо этого, выполняет запись принятых данных в файл, ex_socket_3 — вывод полученных данных на консоль, ex_socket_4 позволяет выбрать файл для записи и записать принятые данные. Вторая программа ex_qe_1 получает от пользователя коэффициенты квадратного уравнения и находит его корни. В качестве модификаций ex_qe_2, ex_qe_3 использованы программы, которые отличаются выводом результата (консоль, файл), а ex_qe_4 содержит ввод значений не через консоль, а через файл.

Таблица 3

Исходные данные эксперимента

Файл	Размер исполняемого кода, кбайт	Общее количество вершин в ГПУ	Количество уникальных вершин
ex_socket_1	86,7	499	192
ex_socket_2	89,7	481	199
ex_socket_3	91,2	489	202
ex_socket_4	89,8	480	205
ex_qe_1	139,2	601	269
ex_qe_2	135,5	616	274
ex_qe_3	135,5	627	278
ex_qe_4	135,9	613	281

Из рис. 9, где представлено дерево памяти классификатора АРТ-2m ГПУ файлов, описанных в табл. 3, видно, что модификации разных программ привели к созданию различных поддеревьев на начальных уровнях. Программы ex_socket_2 и ex_socket_3 схожи по поведению, обе используют потоки для передачи данных, только одна из них использует файловый поток, другая — поток вывода. Результаты проведенного эксперимента показывают, что при схожих размерах файла на качество классификации влияет его содержание. Размеры файлов ex_qe_2 и ex_qe_3 совпадают, но общее количество вершин ГПУ и уникальных вершин различно. Следовательно, размер файла имеет косвенное значение для определения схожести файлов, а основным параметром являются вершины ГПУ.

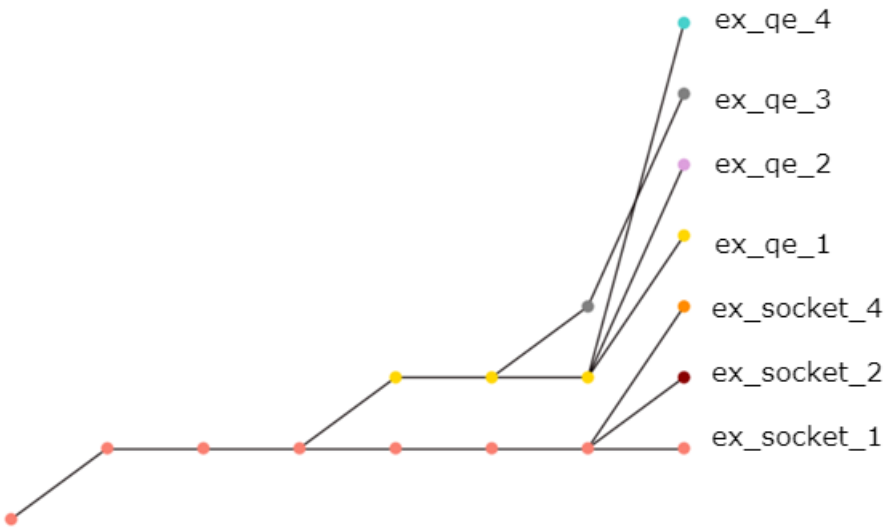

Рис. 9. Дерево памяти классификатора АРТ2-m

7. Исследование временных характеристик при определении ВПО

Анализ времени, затрачиваемого на векторизацию, представлен на рис. 10. Для эксперимента было взято пять файлов размеров 100 кбайт (содержит 4458 инструкций), 200 (9361), 300 (13898), 500 (23446) и 1000 кбайт (46234 инструкций).

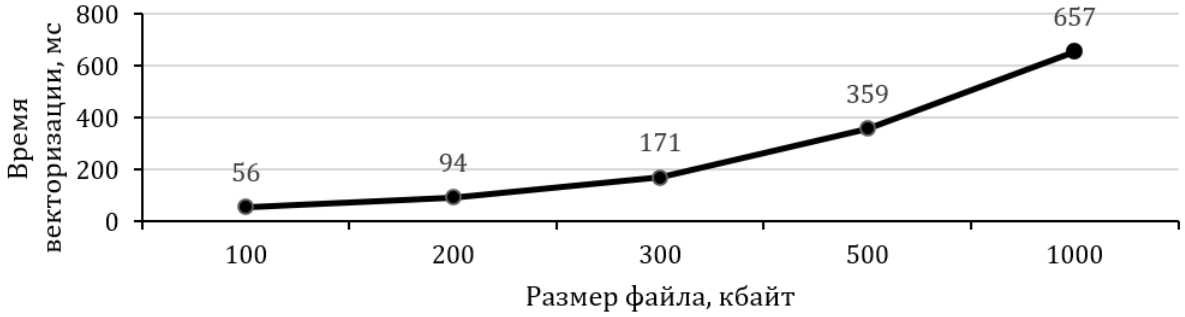

Рис. 10. Время, затрачиваемое на векторизацию файлов различных размеров

Для анализа скорости распознавания образов на вход сети были поданы $m = 100, 500, 1000, 1500, 2000, 2500$ уникальных образов, имеющих 100 входов. В ходе эксперимента использовано 10 образов, номера N_i которых вычисляются по формуле

$$N_1 = \triangle, \quad N_{i+1} = N_i + \triangle, \quad i = 1, \ldots, 9, \quad \triangle = m/10.$$

Среднее время распознавания образа представлено на рис. 11.

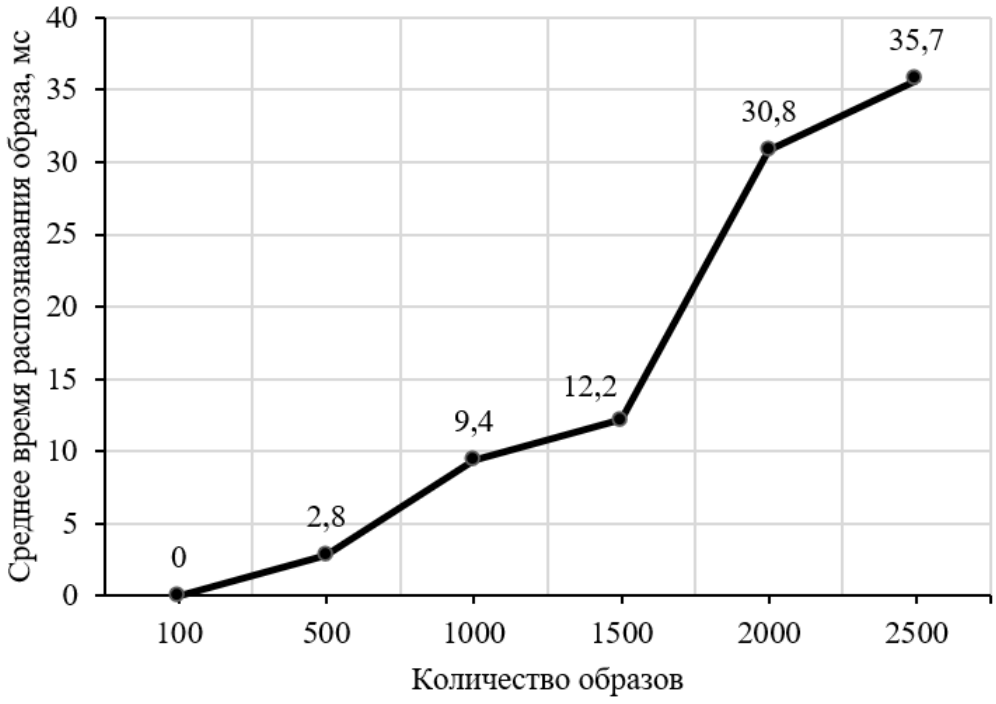

Рис. 11. Среднее время распознавания образа

Графики наглядно демонстрируют быстродействие как модуля программного обеспечения, направленного на векторизацию ГПУ исполняемого кода, так и модуля, реализующего ART-2m. Из результатов видно, что время на анализ и векторизацию исполняемого файла значительно превышает время на классификацию.

Заключение

Проведённые исследования показали, что применение ИНС АРТ-2m для обнаружения ВПО, а также выявления схожести между сохранёнными образами и новыми

даёт высокие результаты: распознавание или добавление нейрона происходит в среднем за 18 мс за счёт древовидной организации памяти. Благодаря использованию ИНС рассмотренной архитектуры можно выявлять схожесть между образами, поданными на вход сети, и затем использовать эти данные для дальнейшего анализа. Получение нового образа происходит в пределах 1 с.

Подход, используемый в векторизации программы, не учитывает семантические связи между вершинами, поэтому для повышения точности предлагается заменить модуль векторизации вершин на векторизацию рёбер ГПУ.

ЛИТЕРАТУРА

1. `https://opendatasecurity.io/how-much-does-a-cyberattack-cost-companies/` — How much does a cyberattack cost companies. 2017.
2. *Salahdine F. and Kaabouch N.* Social Engineering Attacks: A Survey // Future Internet. 2019. V. 11. `https://www.mdpi.com/1999-5903/11/4/89/htm`
3. `https://www.kaspersky.ru/blog/economics-report-2018/20655/` — Во сколько может обойтись потеря данных. 2018.
4. *Харченко С. С., Давыдова Е. М., Тимченко С. В.* Сигнатурный анализ программного кода // Ползуновский вестник. 2012. № 3. С. 60–64.
5. *Babak B. R., Maslin M., and Suhaimi I.* Camouflage in malware: from encryption to metamorphism // Intern. J. Computer Science and Network Security. 2012. V. 12. P. 74–83.
6. *Cai H., Shao Z., and Vaynberg A.* Certified Self-Modifying Code (extended version & coq implementation). Technical Report YALEU/DCS/TR-1379. 2007.
7. *Wei Y., Zheng Z., and Nirwan A.* Revealing packed malware // IEEE Security & Privacy. 2008. V. 6. No. 5. P. 65–69.
8. *Jacob G., Comparetti P. M., Neugschwandtner M., et al.* A static, packer-agnostic filter to detect similar malware samples // LNCS. 2013. V. 7591. P. 102–122.
9. *Linn C. and Debray S.* Obfuscation of executable code to improve resistance to static disassembly // Proc. CCS'03. Washington, USA, 2003. P. 290–299.
10. *Golovkin M.* Systems and methods for detecting obfuscated malware // Patent U.S. 9087195. 2015.
11. *Solomon I. A., Jatain A, and Bajaj S. B.* Neural network based intrusion detection: State of the art // Proc. Intern. Conf. SUSCOM. Amity University Rajasthan, Jaipur-India, February 26–28, 2019.
12. *Bonfante G., Kaczmarek M., and Marion J.* On abstract computer virology from a recursion theoretic perspective // J. Computer Virology. 2009. V. 5. No. 3. P. 263–270.
13. *Narayanan A., Chandramohan M., Chen L., et al.* Subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. arXiv: 1606.08928. 2016.
14. *Burnap P., French R., Turner F., and Jones K.* Malware classification using self organising feature maps and machine activity data // Computers & Security. 2018. V. 73. P. 399–410.
15. *Ahmed F., Hameed H., Shafiq M. Z., and Farooq M.* Using spatio-temporal information in API calls with machine learning algorithms for malware detection // Proc. AISec'09. Chicago, Illinois, USA, 2009. P. 55–62.
16. *Carpenter G. A. and Grossberg S.* ART 2: self-organization of stable category recognition codes for analog input patterns // Appl. Opt. 1987. V. 26. No. 23. P. 4919–4930.
17. *Курмангалеев Ш. Ф., Долгорукова К. Ю., Савченко В. В. и др.* О методах деобфускации программ // Труды Института системного программирования РАН. 2013. Т. 24. С. 145–160.

18. *Буханов Д. Г., Поляков В. М.* Сеть адаптивно-резонансной теории с многоуровневой памятью // Научные ведомости БелГУ. 2018. Т. 45. № 4. С. 709–717.

REFERENCES

1. `https://opendatasecurity.io/how-much-does-a-cyberattack-cost-companies/` — How much does a cyberattack cost companies. 2017.
2. *Salahdine F. and Kaabouch N.* Social Engineering Attacks: A Survey. Future Internet, 2019, vol. 11. `https://www.mdpi.com/1999-5903/11/4/89/htm`
3. `https://www.kaspersky.ru/blog/economics-report-2018/20655/.` 2018.
4. *Harchenko S. S., Davydova E. M., and Timchenko S. V.* Signaturnyy analiz programmnogo koda [Signature analysis of program code]. Polzunovskiy Vestnik, 2012, vol. 3, pp. 60–64. (in Russian)
5. *Babak B. R., Maslin M., and Suhaimi I.* Camouflage in malware: from encryption to metamorphism. Intern. J. Computer Science and Network Security, 2012, vol. 12, pp. 74–83.
6. *Cai H., Shao Z., and Vaynberg A.* Certified Self-Modifying Code (extended version & coq implementation). Technical Report YALEU/DCS/TR-1379. 2007.
7. *Wei Y., Zheng Z., and Nirwan A.* Revealing Packed Malware // IEEE Security & Privacy, 2008, vol. 6, no. 5, pp. 65–69.
8. *Jacob G., Comparetti P. M., Neugschwandtner M., et al.* A static, packer-agnostic filter to detect similar malware samples. LNCS, 2013, vol. 7591, pp. 102–122.
9. *Linn C. and Debray S.* Obfuscation of executable code to improve resistance to static disassembly. Proc. CCS'03, Washington, USA, 2003, pp. 290–299.
10. *Golovkin M.* Systems and methods for detecting obfuscated malware. Patent U.S. 9087195. 2015.
11. *Solomon I. A., Jatain A, and Bajaj S. B.* Neural network based intrusion detection: State of the art. Proc. Intern. Conf. SUSCOM, Amity University Rajasthan, Jaipur-India, February 26–28, 2019.
12. *Bonfante G., Kaczmarek M., and Marion J.* On abstract computer virology from a recursion theoretic perspective. J. Computer Virology, 2009, vol. 5, no. 3, pp. 263–270.
13. *Narayanan A., Chandramohan M., Chen L., et al.* Subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. arXiv: 1606.08928. 2016.
14. *Burnap P., French R., Turner F., and Jones K.* Malware classification using self organising feature maps and machine activity data. Computers & Security, 2018, vol. 73, pp. 399–410.
15. *Ahmed F., Hameed H., Shafiq M. Z., and Farooq M.* Using spatio-temporal information in API calls with machine learning algorithms for malware detection. Proc. AISec'09, Chicago, Illinois, USA, 2009, pp. 55–62.
16. *Carpenter G. A. and Grossberg S.* ART 2: self-organization of stable category recognition codes for analog input patterns. Appl. Opt., 1987, vol. 26, no. 23, pp. 4919–4930.
17. *Kurmangaleev SH. F., Dolgorukova K. YU., Savchenko V. V., et al.* O metodah deobfuskacii programm [About methods of programs deobfuscation]. Proc. Ivannikov Institute for System Programming of the RAS, 2013, vol. 24, pp. 145–160. (in Russian)
18. *Bukhanov D. G. and Polyakov V. M.* Set' adaptivno-rezonansnoy teorii s mnogourovnevoy pamyat'yu [Adaptive resonance theory network with multilevel memory]. Nauchnye Vedomosti BelSU, 2018, vol. 45, no. 4, pp. 709–717. (in Russian)