УДК 004.855.5

© 2020 г. **В.В. Воронин**, д-р техн. наук,
А.В. Морозов
(Тихоокеанский государственный университет, Хабаровск)

МЕТОДИКА КОНТРОЛЯ УГРОЗ БЕЗОПАСНОСТИ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В статье рассмотрены методы обработки данных и построения архитектур нейронных сетей для выполнения поведенческого анализа вредоносного программного обеспечения. Выполнено исследование методики обнаружения угроз безопасности на основе полученных архитектур нейронных сетей. Сделаны выводы о требованиях к данным в рамках рассматриваемой задачи и об эффективности предложенной методики.

Ключевые слова: угрозы безопасности**,** поведенческий анализ, API вызовы, вредоносное программное обеспечение, нейронная сеть, машинное обучение, LSTM, CNN.

DOI: 10.22250/isu.2020.65.3-13

Введение

Стремительное развитие информационных технологий в различных сферах деятельности создает благоприятную среду для разнообразных мошеннических схем, одной из которых является вредоносное программное обеспечение (ПО), разрабатываемое для получения прибыли его создателем. Функциональное назначение такого ПО – нанести определенный вред персональной или корпоративной компьютерной системе. А это в свою очередь обусловливает необходимость создания и внедрения защитных средств от вредоносного ПО.

Однако развитие вредоносного программного обеспечения не стоит на месте, схемы его функционирования постоянно совершенствуются, и, кроме того, возникают принципиально новые образцы. Такая изменчивость приводит к тому, что классические методы обнаружения угроз безопасности уже не обеспечивают требуемого уровня эффективности.

В настоящее время одним из перспективных направлений в области антивирусного ПО считаются технологии на основе поведенческого анализа [1]. В основе этих технологий лежит анализ действий, производимых ПО в вычислительной системе. Ведь программа не может ничего не делать, если бы подобное происходило, то такая программа была бы в той же мере безвредна, сколь и бесполезна. А значит вредоносное ПО вынуждено выполнять действия, преследуя ограниченный набор целей, но в операционной системе ПО может произвести только ограниченный набор действий, следовательно, такие действия можно обнаружить, а их совокупности идентифицировать.

Постановка задачи

Для постановки задачи в рамках данной работы в первую очередь необходимо установить, что такое «поведение» и определить область применения данного термина в отношении вредоносного программного обеспечения.

В общем смысле поведение определяется как «система взаимосвязанных действий, осуществляемых субъектом с целью реализации определенной функции и требующих его взаимодействия со средой».

Когда речь идет о программном обеспечении, в рамках данного определения естественным образом программу следует считать *субъектом* действий.

В свою очередь любая программа производит определенные *взаимосвязанные действия,* определяемые программным кодом. Например, программа может выполнить действие «*А*», в результате которого будет получена информация о местоположении файла в файловой системе, после выполнить действия «*Б*», в результате которого будет произведено воздействие на файл, информация о котором получена благодаря действию «*А*».

Подобные действия производятся с определенной целью, заданной разработчиком этой программы. Так, целью программы в примере, указанном выше, может быть модификация файла для встраивания в его структуру дополнительного кода. Такой механизм работы характерен для вредоносного ПО класса «*Вирус*».

Средой, где производятся действия, в данном случае выступает операционная система, под управлением которой выполняются программы. Операционные системы представляют программам интерфейс для взаимодействия с собой. В случае ОС *Windows* это *API* вызовы. Так, в примере, приведенном выше, в случае выполнения в ОС *Windows* действием «*А*» может выступать *API* вызов «*FindFirstFileA*» [2].

Во время работы программы возможна запись и последующий анализ выполняемых этой программой действий. Так как программа должна выполнять действия последовательно, получая информацию от результата выполнения предыдущего действия, то возможно формирование последовательностей выполняемых ею действий. Эти последовательности имеют распределение во времени и внутренние связи между выполняемыми действиями. Следовательно, для таких последовательностей возможно использование методов *NLP* (*Natural Language Processing*) с применением аппарата нейронных сетей.

Основной задачей данной работы является установление возможности применения машинного обучения для анализа поведения вредоносного ПО и определение эффективной архитектуры нейронной сети для выполнения поведенческого анализа на основе *API* вызовов. При этом необходимо предварительно решить ряд частных задач, а именно: получить и предварительно обработать исходные данные для поведенческого анализа; разработать методику обнаружения угроз безопасности и организовать машинные эксперименты по обнаружению угроз безопасности по данной методике.

Обработка анализируемых данных

Для решения задач машинного обучения необходимы данные, причем предпочтительны большие их объемы, в таком случае становится возможным всесторонне охватить анализируемые данные и выявить в них закономерности.

Далее под образцом понимается пара, состоящая из последовательности *API* вызовов программы и обозначения этой программы.

Последовательности *API* вызовов представляют строки, в которых каждое слово является именем вызванной анализируемой программой функции в той последовательности, в которой эти функции были вызваны от момента старта программы до завершения ее работы. Приведем сокращенный пример такой последовательности: *LdrLoadDll LdrGetProcedureAddress LdrGetProcedureAddress*….

Длина такой последовательности во многом зависит от времени работы программы, эффективности написанного кода и может варьировать от нескольких сотен до многих сотен тысяч вызовов. Обозначение программы представляет из себя название типа программы. Далее для обозначения невредоносного ПО использовано обозначение *NotMalware*, а для обозначения вредоносного ПО – принадлежность конкретной вредоносной программы к определенному типу. Все типы, использованные в данной работе, пред-

ставлены в табл. 1. В качестве генеральной совокупности использованы данные более 25000 образцов различного программного обеспечения, которые также приведены в табл. 1.

Таблица 1

Название	Количество образцов
NotMalware	8474
Virus	6291
Trojan	3489
Dropper	2602
Backdoor	2517
Worms	2102
Downloader	2002
Spyware	1664
Miner	1196
Adware	758
Ransom	502

Часть образцов из табл. 1 была удалена по следующим причинам: неизвестное происхождение или ненадежные образцы; слишком большое количество образцов в выборке; недостаточное количество образца в выборке.

Из итоговой выборки сформированы две рабочие выборки – обучающая, на которой непосредственно производится обучение нейронных сетей, и тестовая для оценки качества методики поведенческого анализа.

Деление на обучающую и тестовую выборки проведено случайным образом, соотношение количества образцов в обучающей и тестовой выборке – 67% и 33% от итоговой выборки соответственно (табл. 2).

Таблица 2

Название	Обучающая выборка	Тестовая выборка
NotMalware	5246	2625
Virus	2876	1413
Trojan	1650	838
Backdoor	1016	500
Miner	815	382
Worms	719	381
Dropper	518	302
Ransom	348	154

В результате содержательного анализа из генеральной выборки выделено 349 уникальных *API* вызовов. Однако, многие *API* вызовы имеют схожие функции при разном именовании. Кроме того, по производимому воздействию функции многих *API* вызовов схожи. Каждый из 349 уникальных *API* вызовов можно отнести к одной из 33 выделенных категорий. Категории

представляют собой сгруппированные по методу воздействия *API* вызовы. Это, например, такие категории как: создание файла; запись в файл; чтение реестра; создание процесса; открытие, чтение в сети; системная информация о пользователе и метрики и др.

Далее, после обработки при помощи полученного словаря из *API* последовательностей образцов были сформированы последовательности пар действий, совершаемых анализируемым ПО. Для таких пар также был сформирован словарь размером 33^2 = 1089 элементов. Данные пары называют N-граммами, и в данном случае использовано значение N = 2. Следовательно, использовались биграммы.

Использование биграмм позволяет сравнивать между собой пары действий и более эффективно устанавливать сходство между анализируемыми действиями.

Методика обнаружения угроз безопасности

Предлагаемая методика построена на основе аппарата нейронных сетей. В ней в качестве базовой архитектуры использованы два варианта архитектур [3]: сверточная нейронная сеть (*CNN*) и нейронная сеть долгой краткосрочной памяти (*LSTM*). Причины выбора первой архитектуры состоят в том, что отличительная способность таких сетей – выделение определенных деталей или в данном случае совокупностей действий. При помощи комбинации скользящих фильтров различной длины возможно проанализировать разные в количественном отношении совокупности биграмм. Аналогично, с выделением деталей изображения, возможно выделение ключевых деталей последовательности действий анализируемого ПО.

Второй тип сети был выбран по причине изначальной предрасположенности таких сетей к анализу последовательностей событий. Каждое последующее действие программы или пара таких действий имеют зависимость от уже выполненных предыдущих действий. Определенные совокупности таких последовательностей могут отличить вредоносное ПО от невредоносного. При помощи сети с долгой краткосрочной памятью можно выявить зависимости данных при их временном разделении, что подходит для решения данной задачи.

В сверточной нейронной сети использованы различные виды слоев, ниже они перечислены, и приведено краткое функциональное назначение каждого слоя и их взаимозависимости.

Conv1D – слой, предназначенный для анализа последовательностей, имеет одно измерение. Во всех слоях модели подобного типа используется

padding со значением *causal*, предназначенный для анализа временных последовательностей, и активационная функция *ReLU*.

Concatenate – слой конкатенации, предназначен для произведения операции конкатенации над выходами слоев параллельной модели.

MaxPooling1D – слой выборки, использующий функцию выбора *Max*().

Flatten – изменяет выходную размерность канала, не влияя на размер партии.

Dense – полносвязный слой, применяется активационная функция *ReLu*.

Dropout – случайным образом устанавливает элементы в ноль, чтобы предотвратить переоснащение.

Dense – выходной слой с активационной функций *softmax*.

Для сети долгой краткосрочной памяти характерен другой набор слоев, перечислены эти слои и дано их краткое описание.

Embedding – входной слой, предназначен для обработки текста, векторизует входные последовательности на основе словаря.

Bidirectional (*LSTM()*) – слой долгой краткосрочной памяти с двунаправленной оберткой, использование двунаправленного LSTM позволяет эффективнее предполагать будущие события [4].

Dense – полносвязный слой, применяется активационная функция *ReLu*.

Dense – выходной слой с активационной функций *softmax*.

Для оптимизации градиентного спуска в предлагаемой методике выбран алгоритм *Adagrad*, который эффективно подстраивает скорость обучения для параметров с низкой частотой появления [5]. В качестве функции вычисления потерь используется функция *sparse_categorical_crossentropy*.

В соответствии с предлагаемой методикой при обучении данные в сеть подаются пакетами различного размера:, 64 единицы для сетей долгой краткосрочной памяти и 16 – для сверточных сетей. Как средство борьбы с переобучением сетей использован метод ранней остановки.

Структура *CNN* состоит из повторяющихся блоков, представленных на рис. 1.

Структура итоговой *CNN* имеет три таких блока, расположенных последовательно. Каждый блок состоит из группы сверточных слоев типа *Conv1D*, расположенных параллельно, за ними следует слой *Concatenate.* Последний слой блока имеет тип *MaxPooling1D и* является входом для следующего блока. После группы блоков следует слой типа *Flatten* и два *Dense* слоя, чередующиеся с *Dropout* слоями. Сеть оканчивается еще одним слоем типа *Dense,* этот слой является выходным.

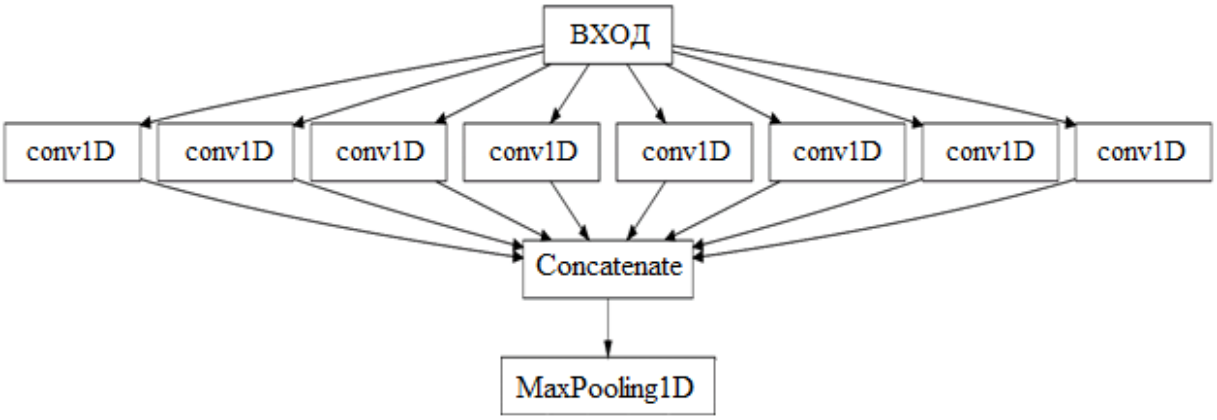

Рис. 1. Блок модели сверточной нейронной сети.

Структура *LSTM* изображена на рис. 2. Размер словаря *Embedding* фиксирован и равен 1089, количество *LSTM* ячеек в *BLSTM* слое равно 256.

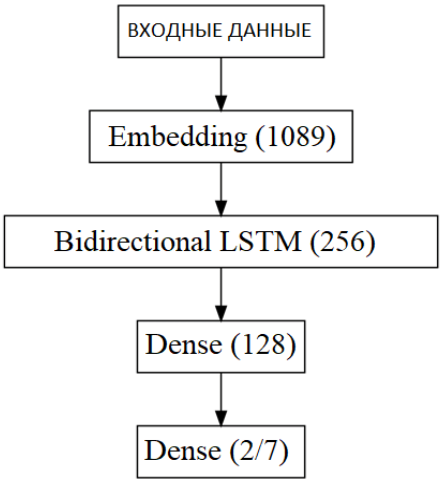

Рис. 2. Модель *LSTM* сети.

Машинные эксперименты по обнаружению угроз безопасности

В данном разделе приведены результаты тестирования полученных архитектур нейронных сетей на тестовой выборке данных, представленной в табл. 2. Модели протестированы при различной длине входящей последовательности – количестве биграмм, поданных на вход нейронной сети.

Для установления точности работы полученных моделей применяется оценка, вычисляемая как отношение числа верно классифицированных образцов к общему число образцов. Результат тестирование *LSTM* сети представлен в табл. 3.

Таблица 3

Тип ПО	Вредоносное ПО			Невредоносное ПО		
Длинна входящей последовательности	500	1500	2000	500	1500	2000
Верно, распознано	3852	3756	3803	2285	2438	2420
Ошибочно распознано	118	214	167	340	187	205
Точность по категориям, в %	97.02	94.6	95.79	87.04	92.87	92.19

Общая точность для разных длин последовательностей: 500 – 93%; 1500 – 93.3%; 2000 – 94.3%. Результат тестирования *CNN* сети – в табл. 4.

Таблица 4

Тип ПО	Вредоносное ПО			Невредоносное ПО		
Длина входящей последовательности	500	1500	2000	500	1500	2000
Верно распознано	3686	3680	3684	2286	2354	2315
Ошибочно распознано	284	290	286	339	271	310
Точность по категориям, в %	92.84	92.69	92.79	87.08	89.67	88.19

Общая точность для разных длин последовательностей: 500 – 90.5%; 1500 – 91.5%; 2000 – 90.9%.

Далее, в машинных экспериментах модель типа *LSTM* использовалась для задачи классификации. Результат тестирования *LSTM* сети в задаче классификации (по 7 категориям) образцов вредоносного ПО показан в табл. 5.

Таблица 5

Класс	Точность классификации, %
Ransom	72,72
Virus	89,45
Trojan	40,21
Miner	71,12
Worm	46,07
Dropper	53,31
Backdoor	22,8

Как видно из результатов тестирования, многие категории классифицируются неверно. Для установления категорий, в которых допущены ошибки, была составлена карта ошибок классификации, представленная на рис. 3.

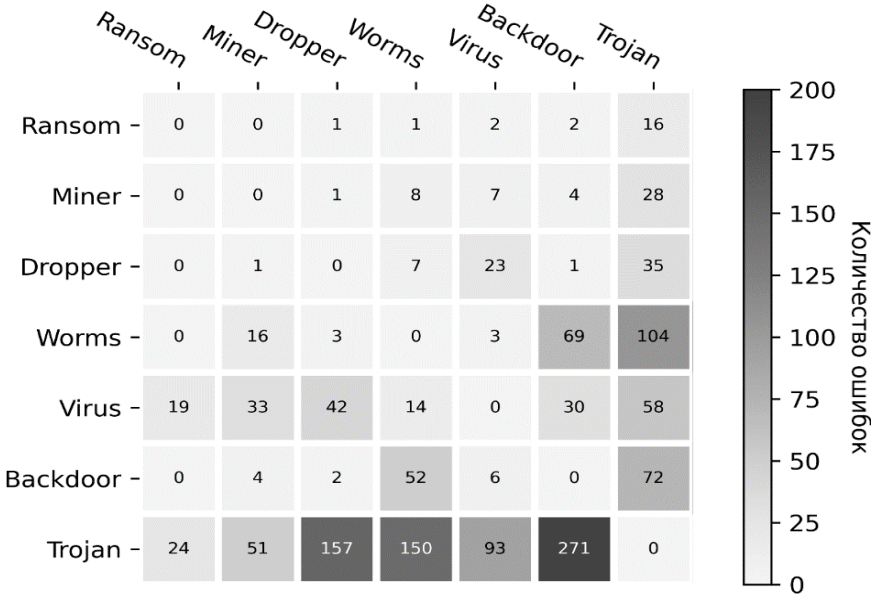

Рис. 3. Карта ошибок модели сети краткосрочной долгосрочной памяти при обучении классификации по 6 категориям.

Из данной карты следует, что наиболее часто допускаются ошибки при сравнении с классом «Trojan». Это происходит из-за того, что класс «Trojan» – имеет достаточно общий характер.

В большинстве представленных классов вероятно пересечение поведенческих паттернов с классом «Trojan» по причине распространенности среди современного вредоносного ПО программ с промежуточным классом. Например, вредоносное ПО WannaCry имеет класс Trojan-Ransom, в его поведении можно обнаружить поведенческие паттерны обоих классов.

Такое поведение образцов приведет к ухудшению качества обучения. Проведем дополнительной тестирование, исключив категорию «Trojan» из обучающей и тестовой выборок.

Результат тестирование *LSTM* сети при задаче классификации образцов вредоносного ПО по 6 категориям представлен в табл. 6 и на рис. 4.

Таблица 6

Класс	Точность классификации (было), %	Точность классификации (стало), %
Ransom	72,72	74,02
Virus	89,45	92,85
Miner	71,12	74,27
Worm	46,07	50,26
Dropper	53,31	75,82
Backdoor	22,8	62,81

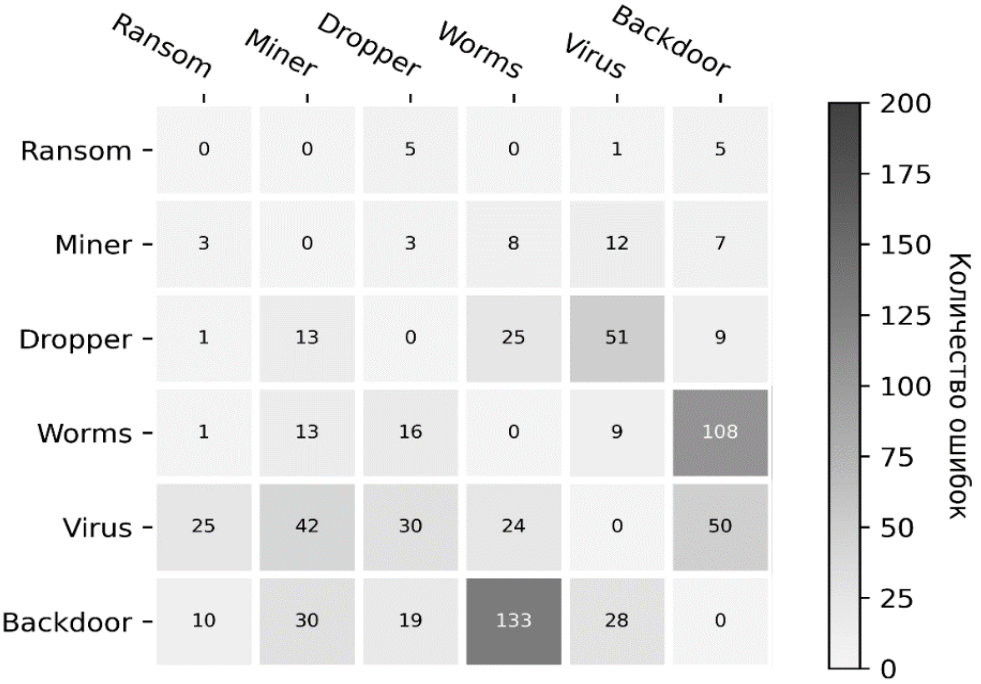

Рис. 4. Карта ошибок модели сети краткосрочной долгосрочной памяти при обучении классификации по 6 категориям.

Результаты тестирования указывают на множественные пересечения паттернов поведения вредоносного ПО, однако эти паттерны отличаются от

паттернов поведения ПО невредоносного. Таким образом, возможно обнаружение вредоносного ПО с высокой точностью, однако задача классификации вредоносного ПО требует указания в базе промежуточных типов вредоносного ПО, в том числе классов пересекающихся, вроде Trojan-Dropper или Worm-Dropper. Наличие такой базы позволит выделить эти образцы вредоносного ПО в отдельную категорию и классифицировать ее.

Заключение

Как можно видеть из полученных результатов тестирования моделей, модель вида *LSTM* показывает лучшие результаты при максимальной достигнутой точности 94,3% против 91.5% в случае модели *CNN*.

При увеличении длины входящей последовательности точность моделей возрастает незначительно, однако увеличение длины входной последовательности вызывает три негативных эффекта, связанных с увеличением:

сложности обучения, так как при увеличении длины последовательности более 2000 символов, для обучения нейронной сети требуется большое количество памяти *GPU*, а также значительно возрастает и время обучения;

времени сбора последовательности, поскольку для сбора длинных последовательностей в системе возрастает время работы анализируемого ПО, а в случае вредоносного ПО этого времени может оказаться достаточно для вредоносного воздействия на систему;

вычислительной сложности при анализе образца вредоносного ПО, которая может негативно сказаться на производительности средств анализа.

Рассмотренная в данной работе методика может стать эффективным инструментом по борьбе с вредоносным программным обеспечением. Так, при достаточном количестве образцов вредоносного ПО, классифицированных по множеству классов, в том числе промежуточных, можно обучить нейронные сети для обнаружения образцов вредоносного ПО и их классификации.

В ходе тестирования и рассмотрения результатов было установлено, что такая архитектура как сеть долгой краткосрочной памяти является эффективной архитектурой для анализа *API* последовательностей вредоносного ПО.

В ходе проведения классификации вредоносного ПО установлены множественные пересечения поведенческих паттернов для ПО различных типов, что свидетельствует о схожих механизмах работы анализируемого ПО.

Дальнейшие исследования будут нацелены на углубленное изучение

поведенческих паттернов вредоносного ПО и связей между ними, а также установление связи между образцами вредоносного ПО в рамках его поведения. Кроме того, планируется исследование функциональных свойств моделей типа *Transformer* [6].

ЛИТЕРАТУРА

1. *Сорокин С.В., Сорокин А.С.* Использование нейросетевых моделей в поведенческом скоринге // Прикладная информатика. – 2015. – № 2 (56). – С. 92-109.
2. Документация по win32 API вызовам [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/windows/win32/api/ – (Дата обращения: 01.07.2020).
3. *Bai S., Kolter J.Z., Koltun V.* An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling [Электронный ресурс]. – 2018. – Режим доступа: https://arxiv.org/pdf/1803.01271.pdf. – (Дата обращения: 01.05.2020).
4. *Ray A., Rajeswar S., Chaudhury S.* Text recognition using deep BLSTM networks // 2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR), Kolkata. – 2015. – P. 1-6
5. *Ruder S.* An overview of gradient descent optimization algorithms [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/1609.04747.pdf. – (Дата обращения: 01.07.2020).
6. *Vaswani A., Shazeer N., Parmar N.* Attention Is All You Need // NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems. – 2017. – P. 6000-6010.

Статья представлена к публикации членом редколлегии С.В. Шалобановым.

E-mail:
Воронин Владимир Викторович – 004183@pnu.edu.ru;
Морозов Алексей Владимирович – 2014102127@pnu.edu.ru.