МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки 02.03.02 — "Фундаментальная
информатика и информационные технологии"

АВТОМАТИЗИРОВАННАЯ КЛАССИФИКАЦИЯ ФУНКЦИОНАЛЬНОГО НАЗНАЧЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПО ИСХОДНЫМ КОДАМ С ИСПОЛЬЗОВАНИЕМ НЕЧЕТКИХ ХЭШ-ФУНКЦИЙ

Выпускная квалификационная работа
на степень бакалавра

Студентки 4 курса
Я. А. Задверняк

Научный руководитель:
к.ф.-м.н, доцент В. А. Нестеренко

Допущено к защите:
руководитель направления ФИИТ _____ В. С. Пилиди

Ростов-на-Дону
2020

ОТЗЫВ

руководителя на выпускную квалификационную работу

бакалавра

Института математики, механики и компьютерных наук им. И.И. Воровича

Южного Федерального Университета

Задверняк Яны Анатольевны

по теме «Автоматизированная классификация функционального назначения программного обеспечения по исходным кодам с использованием нечетких хэш-функций»,

представленной к защите по направлению:

02.03.02 – Фундаментальная информатика и информационные технологии

направленность программы

«Фундаментальная информатика и информационные технологии»

Выпускная квалификационная работа бакалавра Задверняк Яны Анатольевны — «Автоматизированная классификация функционального назначения программного обеспечения по исходным кодам с использованием нечетких хэш-функций» обладает бесспорной актуальностью в сфере информационной безопасности. Студентом тщательно проанализированы и подобраны технологии, методы и инструменты для реализации цели и задач выпускной квалификационной работы.

В ходе исследования Яна Анатольевна рассмотрела имеющиеся реализации машинной классификации, кластеризации, чтобы оценить возможность их применения для анализа исходных кодов с использованием нейронных сетей (в качестве способа автоматизации классификации). Студентом была собрана база кода вредоносного программного обеспечения, осуществлен парсинг выборки для обучения нейронной модели. Для демонстрации результатов Задверняк Я.А. реализовала стенд-приложение.

Во время подготовки ВКР автор на высоком уровне продемонстрировала полученные при обучении компетенции. Деятельность Яны Анатольевны, направленная на написание работы бакалавра, была своевременной, аккуратной, с большой долей инициативности и вовлеченности.

Считаю, что с поставленными задачами Задверняк Яна Анатольевна справилась, выпускная квалификационная работа отвечает всем требованиям, предъявляемым к выпускным квалификационным работам, а автору может быть присвоена квалификация «бакалавр» по направлению 02.03.02 – Фундаментальная информатика и информационные технологии.

Считаю, что Задверняк Яна Анатольевна по результатам проделанной работы заслуживает оценку — «отлично».

к.ф.-м.н., научный руководитель  /В.А. Нестеренко/

_____
(подпись, дата)

«_17_» _июня_____ 2020г.

Ознакомлен  _____ /Я.А. Задверняк/

(подпись, дата)

«_17_» _____июня_____ 2020г.

Задание на выпускную квалификационную работу

студентки 4-го года бакалавриата Задверняк Я.А.

Направление подготовки: 02.03.02 – Фундаментальная информатика и информационные технологии, направленность программы «Фундаментальная информатика и информационные технологии»

Студент: Я.А. Задверняк

Научный руководитель: доц., к. ф.-м. н., В. А. Нестеренко

Год защиты: 2020

Тема работы: Автоматизированная классификация функционального назначения программного обеспечения по исходным кодам с использованием нечетких хэш-функций.

Цель работы: Исследовать возможности использования нейронных сетей для классификации исходных кодов вредоносных программ.

Задачи работы:

- проанализировать существующие решения в сфере классификации программ с использованием нейросетей;
- осуществить сбор обучающей выборки из исходных кодов вредоносных программ разных классов;
- обучить нейронную сеть на различных метриках абстрактных синтаксических деревьев;
- проанализировать полученные результаты обучения.

Научный руководитель ,
доц., к. ф.-м. н. _____ В.А. Нестеренко

Студент 4-го курса
бакалавриата _____ Я.А. Задверняк

25.11.2019

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Южный Федеральный Университет

СПРАВКА
о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе Антиплагиат.ВУЗ

Автор работы	**Задверняк Яна Анатольевна**
Подразделение	кафедра Информатики и ВЭ
Тип работы	Выпускная квалификационная работа
Название работы	Автоматизированная классификация функционального назначения программного обеспечения по исходным кодам с использованием нечетких хэш-функций
Название файла	диплом.pdf
Процент заимствования	**1.10 %**
Процент самоцитирования	**0.00 %**
Процент цитирования	**10.01 %**
Процент оригинальности	**88.89 %**
Дата проверки	**12:24:54 17 июня 2020г.**
Модули поиска	Модуль поиска ИПС "Адилет"; Модуль выделения библиографических записей; Сводная коллекция ЭБС; Модуль поиска "Интернет Плюс"; Коллекция РГБ; Цитирование; Модуль поиска переводных заимствований; Модуль поиска переводных заимствований по elibrary (EnRu); Модуль поиска переводных заимствований по интернет (EnRu); Коллекция eLIBRARY.RU; Коллекция ГАРАНТ; Коллекция Медицина; Диссертации и авторефераты НББ; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска перефразирований Интернет; Коллекция Патенты; Модуль поиска "ЮФУ"; Модуль поиска общеупотребительных выражений; Кольцо вузов
Работу проверил	Нестеренко Виктор Александрович ФИО проверяющего
Дата подписи	Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Содержание

Введение

Целью данной научной работы являлось исследование возможности использования нейронных сетей для классификации исходных кодов вредоносных программ. Для достижения цели были поставлены следующие задачи:

- анализ существующих решений в сфере классификации программ с использованием нейросетей
- сбор обучающей выборки из исходных кодов вредоносных программ разных классов
- обучение нейронной сети на различных метриках абстрактных синтаксических деревьев
- анализ результатов обучения

1. Введение в предметную область

По состоянию на март 2020 года JavaScript используется на 90% всех сайтов в интернете [1]. Но такая скорость развития веба несет в себе угрозу безопасности, поскольку помимо количества клиентского кода на JavaScript выросло и количество вредоносного кода на этом языке.

Согласно статистике, предоставляемой «Лабораторией Касперского», за последний год разработанный ими антивирус обнаружил более 20 миллионов уникальных вредоносных элементов на различных сайтах, среди которых криптомайнеры, загрузчики вредоносных программ, эксплуатации эксплойтов в безопасности [2]. Важно понимать, что большинство вредоносных скриптов написаны именно на JavaScript, как на самом популярном скриптовом языке в вебе.

У вредоносных скриптов на JavaScript есть несколько особенностей, делающих их ощутимой проблемой. Во-первых, поскольку все

современные браузеры являются интерпретаторами JavaScript, код на нем активно используется в XSS-атаках. В этом случае, с использованием уязвимостей код внедряется на страницу и исполняется на компьютере пользователя, выполняя необходимые для злоумышленника действия. Несмотря на то, что сейчас существует много методов защиты от таких атак, XSS все равно входит в OWASP Top-10 и располагается там на 7 месте [3].

Во-вторых, код на JavaScript легко обфусцируется. Существует много программ-обфускаторов, производящих обфускацию разной степени сложности. Несмотря на то, что одна из изначальных сфер ее применения – защита исходного кода проприетарного ПО, очень широко распространено ее использование для сокрытия вредоносных скриптов от антивирусов.

Чтобы понять, как обфускация мешает обнаружению скриптов, стоит обратиться к методам обнаружения вредоносного ПО. Сейчас существует два основных метода обнаружения: сигнатурный и основанный на поведении.

Сигнатурный метод основан на проверке сигнатуры программы и сопоставлении ее с образцами заведомо вредоносных программ. Большое количество обфускаторов делает этот метод неэффективным, поскольку при обфускации сигнатура программы меняется в соответствии с обфускатором, и сигнатурным методом все изменения отследить невозможно.

Метод, основанный на поведении, отслеживает деятельность программы и обнаруживает совершаемые ей подозрительные действия. Этот метод устойчив к обфускации, но требует много времени и ресурсов. Иногда проверка сайта методом, основанном на поведении, требует тщательной симуляции действий пользователя на этом сайте, вплоть до прокрутки страницы со скоростью, свойственной человеку, и симуляции движения курсора. Такая проверка может занимать довольно много реального времени. Таким образом, этот метод не лучшим образом подходит для использования в вебе, поскольку

для веб-страниц важной является скорость загрузки, а проверка всех расположенных на них скриптов может занять много времени.

Отсюда следует, что оба метода имеют существенные недостатки в случае необходимости обнаружения вредоносных скриптов на веб-странице.

По результатам предыдущей работы эмпирическим образом был подобран метод обнаружения вредоносных скриптов с помощью абстрактных синтаксических деревьев, построенных на исходном коде. Было выдвинуто предположение о том, что АСД вредоносных программ имеют схожие элементы, что было подтверждено экспериментом. Предложенный метод хорошо показал себя для обнаружения вредоносных программ определенного класса – в данном случае, обфусцированных скриптов. Но таким образом нельзя выявить новые классы угроз, что является большим минусом для метода обнаружения вредоносного ПО.

В последнее время сфера машинного обучения расширилась, и теперь включает в себя такие задачи, как классификация и кластеризация текста, сентиментальный анализ, отфильтровывание спама и поиск в интернете. Методы, которые применяются в этих сферах, можно применять и для анализа исходных кодов.

Поэтому для улучшения эффективности работы метода обнаружения вредоносных программ по их АСД, рассмотрим автоматическую классификацию скриптов с использованием нейронных сетей, где в качестве исходных данных будет использоваться исходный код.

2. Существующие решения применения нейронных сетей для классификации программ

Большое количество open-source проектов и исходных кодов в открытых источниках дает возможность собрать большую базу для обучения нейронных сетей анализу содержимого программ.

Автоматический анализ исходного кода может быть очень полезен для поиска ошибок и уязвимостей, а также для представления фрагментов кода для их дальнейшего исследования.

Существует много способов проанализировать различные характеристики программы по их исходному коду. При этом используются различные методы: анализ числовых метрик, лексем и даже комментариев на естественном языке. Каждый метод выбирается в соответствии с целью, которую преследуют разработчики нейронных сетей.

В задаче классификации программ неожиданные применения находят методы обработки естественного языка (Natural Language Processing или NLP). Глобально целью NLP является обучение машин пониманию нашей устной или письменной речи. NLP включает в себя много задач, среди которых выделение токенов и классификация текстов или предложений. [4] И хоть языки программирования во многом далеки от естественных, есть примеры того, как методы решения задач NLP использовались для анализа и классификации исходных кодов.

Так, например, анализ естественных языков в чистом виде хорошо подходит для решения проблем с самопризнанным техническим долгом. Самопризнанный технический долг возникает в тех случаях, когда разработчик решает пожертвовать особенностями требований или дизайна для быстрого решения появившейся перед ним проблемы. От обычного технического долга он отличается тем, что возникает в результате осознанного решения разработчика. Соответственно, места его возникновения должны документироваться. Для определения типа долга используются отфильтрованные комментарии, их тип (однострочные, многострочные или документация), их расположение в коде и их контекст (в какой из структурных единиц кода они располагаются). Затем используются методы анализа естественных языков для определения содержания комментариев. В итоге задача сводится к обычной задаче обработки естественных языков. Согласно резуль-

татам анализа, блоки кода, выделенные с помощью анализа комментариев, делятся на два класса - код, не содержащий в себе признаков самопризнанного технического долга и код, являющийся техническим долгом. [5]

Существуют и исследования о возможности использования нейронных сетей для поиска в проектах кода, который с большой вероятностью является признаком уже существующих или потенциальных проблем в системе. В этом случае данными для обработки являются числовые метрики исходного кода, полученные другими анализаторами. По этим исходным данным нейросеть распознает, к какому классу принадлежит код. [6]

Помимо кода, являющегося признаком потенциальных уязвимостей, можно также использовать нейросети для того, чтобы находить код с уязвимостями. Для этого можно анализировать программы на функциональном уровне, поскольку это - самый низкий уровень, позволяющий отследить поток выполнения подпрограмм. На этом уровне специально написанный лексер выделяет минимально возможное для сохранения логики работы программы количество токенов. Нейросеть обучается на выделенных из программ токенов, разделенных на два класса: “Уязвимые” и “Без уязвимостей”. В этом случае для обучения также используются некоторые подходы, которые используют при обучении нейросетей, классифицирующих единицы естественного языка. [7]

Одним из методов, использующихся в NLP является doc2vec, позволяющий переводить текст произвольной длины в числовое представление. Этот метод тоже нашел свое применение в анализе вредоносности скриптов. Его можно использовать, например, для определения JavaScript-атак. В этом случае делается предположение, что так как разработанный для обмана пользователя код на контекстном уровне отличается от другого, то в нем используются некоторые характерные “вымогательные” слова. Если проанализировать базы вредоносных кодов, то можно прийти к выводу, что существуют

слова в названиях переменных и функций вредоносных кодов, которые используются чаще, чем другие. Для определения встроенного вредоносного кода используется представление в виде абстрактного синтаксического дерева, которое и подается на вход методу doc2vec.В итоге, с помощью этого метода по контекстному признаку происходит классификация кода как несущего или не несущего угрозы. [8]

Еще один пример использования абстрактного синтаксического дерева программы для классификации исходных кодов - нейросеть, обрабатывающая образованные из синтаксических деревьев графы. Модель обучается таким образом, чтобы она могла аппроксимировать функции перехода между графом и скрытыми переменными (содержащими в себе специфически сжатые данные о входном элементе [9]). Затем полученная скрытая переменная уже исследуется с целью классификации соответствующего ей графа и, соответственно, исходного кода. [10]

Близко к задаче классификации стоит задача генерации фрагментов кода, реализующих логику какого-либо класса программ. Для этого могут использоваться и представления, полученные в результате работы классификатора. Например, полученные представления абстрактных синтаксических деревьев в виде скрытых переменных можно использовать для промежуточного этапа генерации какого-либо фрагмента кода, который будет основан на структуре этого дерева. [10]

Можно сделать вывод, что нейронные сети имеют широкое применение в сфере классификации исходных кодов. При этом на этапе подготовки входных данных и при обучении нейросети используются разные методы. Часто на каком-то из этапов обучения модели фигурируют абстрактные синтаксические деревья деревья, либо как входные данные для нейронной сети, либо как промежуточное представление.

Используем нейронные сети для улучшения нашего метода определения вредоносных программ по их абстрактному синтакси-

ческому дереву, научив модель классифицировать исходные коды по классу угрозы.

3. Практический эксперимент

Для того, чтобы проверить возможность определения типа вредоносных программ по их абстрактным синтаксическим деревьям был проведен эксперимент. Он проводился в несколько этапов:

- Сбор обучающей выборки
- Парсинг обучающей выборки.
- Извлечение метрик из полученных абстрактных синтаксических деревьев
- Обучение модели на различных комбинациях метрик
- Создание приложения для демонстрации работы обученной модели

3.1. Создание обучающей выборки

В качестве обучающей выборки использовалась база из двух классов вредоносных кодов.

Первый класс - вредоносные криптомайнеры. В последнее время стал популярен майнинг посредством браузера, использующий чистый JavaScript. Этот метод называется криптоджекингом. Злоумышленник использует ресурсы компьютера жертвы для получения выгоды. Вредоносные криптомайнеры исполняются непосредственно браузером и не требуют установки сторонних программ. Такая атака приводит к снижению производительности, загрузке процессора и перегреву. Такому виду атак подвержены и телефоны, в их случае майнинг может приводить к физическим повреждениям батареи

устройства. Несмотря на то, что деятельность криптомайнера имеет ярко выраженные симптомы, перед его обнаружением может пройти много времени, в результате чего могут быть повреждены компоненты компьютера. Таким образом, важной является задача предотвращения запуска сторонних криптомайнеров.[11]

Второй класс - кейлоггеры. Эти скрипты перехватывают и записывают в локальный файл или отправляют на сервер злоумышленника данные о работе клавиатуры. У кейлоггеров много возможностей для применения, но чаще всего они используются для перехвата чувствительной информации пользователя, такой как пароли или платежная информация. Обнаружить скрипт-кейлоггер бывает непросто, поскольку обычно он работает незаметно в качестве фонового процесса. [12]

3.2. Парсинг и обучение модели

Для парсинга исходных текстов скриптов из обучающей выборки использовался парсер Esprima. Из полученных деревьев выделялись следующие метрики: общее количество узлов в дереве, а также относительное количество узлов разных типов. Таким образом, выборка состоит из объектов JSON вида

```
{
"type":2, //класс программы
"length":66, //количество узлов
"varDec":0.03, //узлы типа VariableDeclaration
"callExp":0.075, //узлы типа CallExpression
"assignExp":0.045, //узлы типа AssignmentExpression
"literal":0.09, //узлы типа Literal
"expStat":0.09, //узлы типа ExpressionStatement
"ident":0.378 //узлы типа Identifier
}
```

Для создания и обучения модели было решено использовать open-source библиотеку TensorFlowJS.

У нее есть несколько преимуществ в рамках решаемой задачи.

Во-первых, TensorflowJS включает в себя полноценное API для низкоуровневых компонентов и алгоритмов машинного обучения, что позволяет описывать модели на высоком уровне.

Во-вторых, поскольку с помощью TensorFlowJS модели создаются, обучаются и используются на JavaScript, это дает возможность исполнять их непосредственно в браузере.

В-третьих, эта библиотека позволяет быстро переобучать модель при появлении новых данных, что очень полезно для задачи классификации угроз. Каждая новая обнаруженная угроза с известным классом может использоваться для дообучения модели и повышения точности ее работы.

Основным элементом представления данных в TensorFlow являются тензоры, представляющие собой набор значений примитивного типа, объединенные в массив любой размерности. Сама модель при этом состоит из слоев, которые отвечают за изменение весов и, соответственно, обучение.

Для классификации была создана последовательная модель из трёх полносвязных слоев: входного, скрытого и выходного. Их полносвязность означает, что любой нейрон текущего слоя связан с каждым нейроном следующего слоя. По своей архитектуре созданная модель представляет собой классический многослойный персептрон.

В качестве функции активации для обоих слоев использовалась функция softmax, а в качестве функции потерь - перекрестная энтропия. Это одни из самых популярных компонентов, используемых в сверточных нейронных сетях, но помимо этого являются очень эффективными в задачах, связанных с классификацией. [13]

TensorFlow также дает возможность использовать оптимизаторы, ускоряющие и увеличивающие производительность обучения модели. Для созданной модели был использован оптимизатор Adam со скоростью обучения 0.05. Скорость обучения была подобрана экспериментально. Основной принцип работы этого оптимизатора - вы-

численение индивидуальных значений скорости обучения исходя из различных параметров и применение их в методе стохастического градиентного спуска.

Все данные обучающей выборки нормализовались перед процессом обучения. После того, как полученные в результате парсинга метрики конвертировались в тензор, они нормализовались методом min-max. Он позволяет получить равномерное распределение нормализованных данных в выборке в интервале $[0, 1]$. Нормализация данных повышает эффективность обучения и стабильность модели. [14] Также нормализация очень важна именно в случае обучения моделей с использованием TensorFlow, поскольку большинство моделей обучения внутри библиотеки спроектированы так, чтобы работать с маленькими числами.

В случае нормализации в TensorFlow, мы также работаем с тензорами. Библиотека позволяет использовать все необходимые операторы над тензорами, поэтому нормализация выглядит следующим образом:

```
const max = inputTensor.max()
const min = inputTensor.min()
const normalized = inputTensor.sub(min).div(max.sub(min))
```

Значения `max` и `min` после подготовки выборки сохранялись в отдельные JSON-файлы. Поскольку тензоры представляют собой массивы значений примитивного типа, в файл записывались результаты вызова функции `tensor.arraySync()`, возвращающей простое представление тензора в виде массива. Сохраненные тензоры важны для дальнейшего использования полученной модели.

Экспериментальным путем было доказано, что для большинства входных значений в среднем модель достигает максимальной точности за 50 эпох. Таким образом, во всех дальнейших исследованиях оценивалась точность обученной в течение 50 эпох модели.

Для обучения в качестве входных данных использовались различные комбинации полученных на предыдущем шаге метрик. По-

сле каждой эпохи обучения оценивалась точность модели на тестовой выборке. После 50 эпохи оценивалась максимальная точность, достигаемая моделью.

3.3. Результаты обучения модели

Были получены следующие результаты:

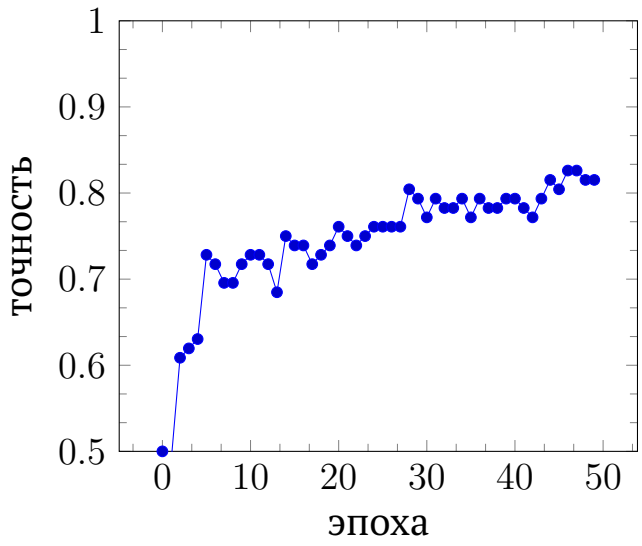

Рисунок 1 — Обучение модели для узлов VariableDeclaration, CallExpression, Literal, AssignmentExpression

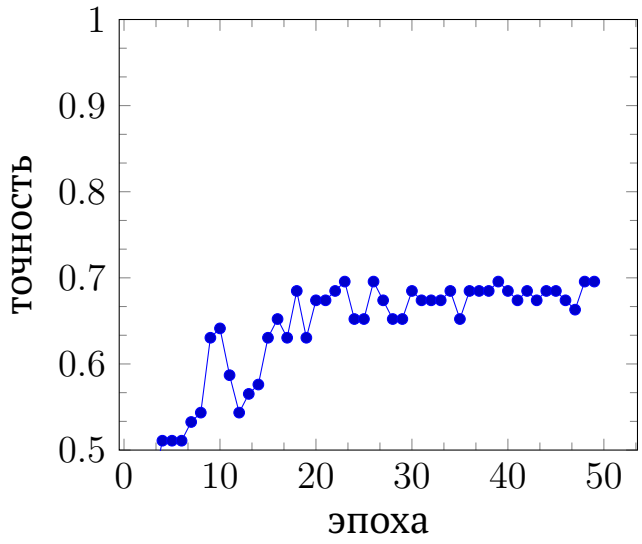

Рисунок 2 — Обучение модели для узлов Identifier, CallExpression, ExpressionStatement, AssignmentExpression

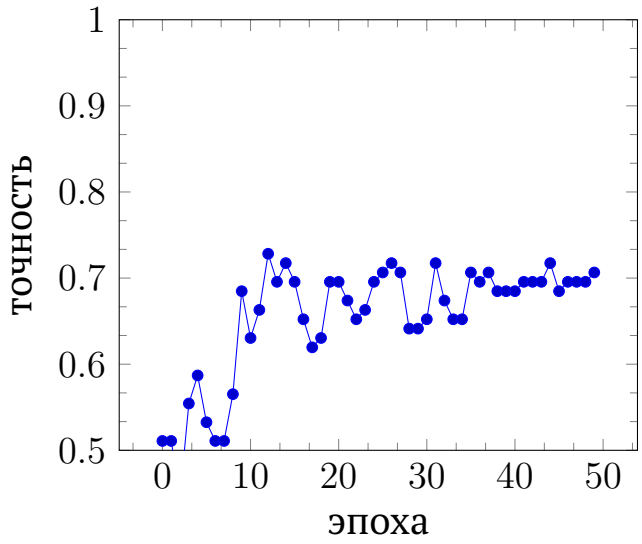

Рисунок 3 — Обучение модели для общего количества узлов и узлов Identifier, VariableDeclaration, CallExpression

Как следует из рис. 2, 3, для большинства комбинаций метрик точность модели по прошествии 50 эпох обучения стремилась к 0.65.

Обучение же модели на количестве узлов типа "объявление переменных "вызов функции "литерал"и "выражение присваивания"показывает намного лучшие результаты, как видно на рис. 1.

Можно предположить, что такой набор метрик является определяющим в задаче классификации криптомайнеров и кейлоггеров. В этом случае уже к 10 эпохе точность достигает 0.7, впоследствии к концу обучения приближаясь 0.8.

Такой показатель точности указывает на то, что по данным значениям, основанным на абстрактных синтаксических деревьях, нейросеть с большой вероятностью отличит криптомайнер от кейлоггера. Это доказывает предположение о том, что существуют классы программ, у АСД которых есть характерные черты, позволяющие их классифицировать.

3.4. Демонстрационное приложение

Модель была обучена на наборе метрик, дающем наибольший результат, и сохранена. Модуль TensorFlow для NodeJS позволяет сохранить модель в виде .json-файла, в котором находится топология, и .bin-файла с весами.

Для демонстрации возможностей полученной модели и ее работоспособности было создано веб-приложение с клиентом, написанным с помощью фреймворка Angular и сервером, написанным на NodeJS. Суть его заключается в следующем:

- Клиент отсылает JavaScript файл на сервер
- Сервер с помощью обученной модели классифицирует полученный скрипт
- Клиент получает результат классификации

При запуске сервера, загружается сохраненная ранее модель. Её формат позволяет использовать уже обученную модель в любом скрипте, при этом предварительная подготовка заключается только в нормализации данных в соответствии с полученными при подготовке выборки параметрами. Для этого и загружаются сохраненные

на этапе обучения модели файлы с минимальным и максимальным тензором обучающей выборки. Только их использование для нормализации произвольных данных позволит модели классифицировать их корректно.

Сервер демонстрационного приложения получает JavaScript-файл от пользователя, считывает его содержимое и использует парсер Esprima для извлечения метрик. Полученные метрики преобразуются в тензор, значения которого нормализуются тем же принципом, что и при подготовке обучающей выборке. Затем результат `model.predict(normalizedData)` отправляется в клиентское приложение. Клиентское приложение, получив результат, выводит его на веб-странице.

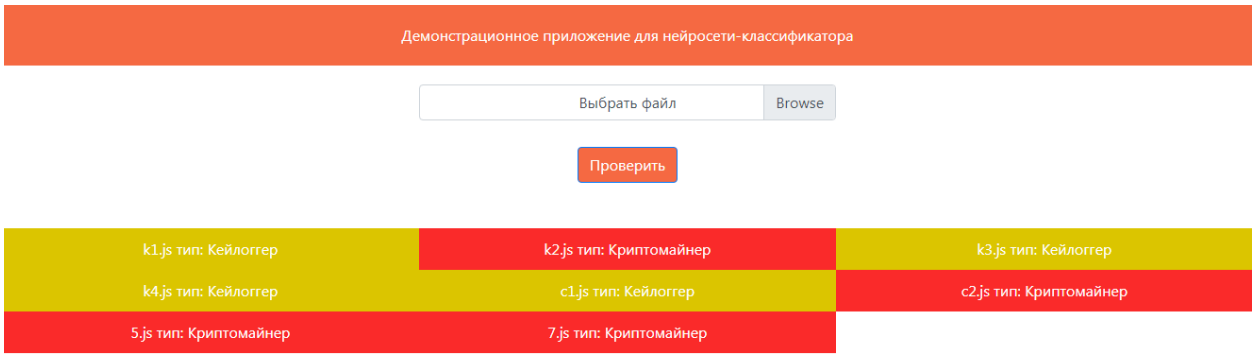

Рисунок 4 — Пример работы нейросети в демонстрационном приложении

Написанное приложение демонстрирует простоту использования модели-классификатора, обученной на метриках АСД. Парсинг исходного кода в АСД - довольно распространенная задача, поэтому есть много методов, ускоряющих этот процесс. [15] Использование уже обученной модели также работает быстро и не является ресурсозатратным. Таким образом, любой проект, имеющий доступ к парсеру Esprima и библиотеке Tensorflow, может легко применять полученную модель, загрузив ее топологию и веса, а также тензоры для нормализации.

4. Возможности развития метода

При подборе более точных и сложных метрик и при увеличении объема обучающей выборки данный метод может показывать более высокую точность при классификации, а также распознавать больше классов вредоносных программ. Также важной особенностью является возможность легко дообучать модель при обнаружении новых угроз.

Подобная модель может использоваться в антивирусах для распознавания угроз и определения классов новых угроз. Поскольку предложенный метод более быстрый и менее трудозатратный, чем динамический метод обнаружения вредоносных программ, его можно использовать для обнаружения интернет-угроз, например, вредоносных скриптов на страницах и нежелательных криптомайнеров.

Данный метод также можно развивать для классификации программ, написанных на разных языках, поскольку он зависит не от конкретного языка, а от построенных на исходных кодах АСД. Для этого необходимо использовать парсеры разных языков, генерирующие деревья с эквивалентной нотацией. Такое развитие позволит существенно ускорить и облегчить определение различных видов вредоносных программ.

Заключение

Можно сделать вывод, что абстрактные синтаксические деревья могут использоваться для классификации вредоносных программ по их назначению.

В результате работы проанализированы существующие решения в сфере классификации программ с использованием нейросетей. На основании собранной выборки обучена модель, классифицирующая криптомайнеры и кейлоггеры с точностью 0.8.

Также было создано приложение, демонстрирующее использование обученной модели.

Исходный код демонстрационного приложения, а также код, использованный для обучения модели, опубликованы в репозитории GitHub: https://github.com/DasIgnis/AstNeuralNetworkClassify.

Список литературы

1. Usage statistics of JavaScript for websites. — URL: `https://w3techs.com/technologies/details/pl-js` (дата обр. 31.05.2020).
2. Kaspersky Security Bulletin 2019. Statistics. — URL: `https://go.kaspersky.com/rs/802-IJN-240/images/KSB_2019_Statistics_EN.pdf` (дата обр. 31.05.2020).
3. OWASP Top Ten. — URL: `https://owasp.org/www-project-top-ten/` (дата обр. 31.05.2020).
4. *Ganegedara T.* Natural Language Processing with TensorFlow. — Birmingham : Packt Publishing Ltd., 2018.
5. *S. Maldonado E. da*, *Shihab E.*, *Tsantalis N.* Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt // IEEE transactions on software engineering. — 2016.
6. *Kim D. K.* Finding Bad Code Smells with Neural Network Models // Computer Science International Journal of Electrical and Computer Engineering. — 2017.
7. *Russell R. L.*, *Kim L.*, *Hamilton L. H.* Automated Vulnerability Detection in Source Code Using Deep Representation Learning // 17th IEEE International Conference on Machine Learning and Applications. — 2018.

8. *Ndichu S.*, *Kim S.*, *Ozawa S.* A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors // Applied Soft Computing Volume 84. — 2019.
9. *Wiewel S.*, *Becher M an Thuerey N.* Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow // Computer Graphics Forum. — 2019.
10. *Vytovtov P.*, *Chuvilin K.* Unsupervised Classifying of Software Source Code Using Graph Neural Networks // 2019 24th Conference of Open Innovations Association (FRUCT). — 2019.
11. Malicious Cryptominers. — URL: `https://www.eset.com/lv-ru/malicious-cryptominers/` (дата обр. 31.05.2020).
12. Что такое Кейлоггер? — URL: `https://www.kaspersky.ru/blog/chto-takoe-keylogger/700/` (дата обр. 31.05.2020).
13. *Liu W.*, *Wen Y.*, *Yu Z.* Large-Margin Softmax Loss for Convolutional Neural Networks // Proceedings of the 33 rd International Conference on Machine Learning. — 2017.
14. Data Preparation and Feature Engineering for Machine Learning. — URL: `https://developers.google.com/machine-learning/data-prep/transform/normalization` (дата обр. 30.05.2019).
15. *Yu Y.* fAST: Flattening Abstract Syntax Trees for Efficiency // 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). — 2019.