Искусственный интеллект в кибербезопасности: поиск вредоносного программного обеспечения

Д.Е. Намиот, Е.А. Ильюшин

***Аннотация*— В настоящей статье мы подробно рассматриваем технические аспекты поиска (классификации) вредоносного программного обеспечения с помощью машинного (глубокого) обучения. Рассмотрен как классический пример классификации с использование традиционных моделей типа многослойного перцептрона, SVM, Random Forest и т.п., так и подходы, связанные с использованием больших языковых моделей. Основная проблема в такого рода задачах лежит в области подготовки и сбора данных. Анализ кода на предмет поиска вредоносного программного обеспечения бывает статический, динамический (во время выполнения) и гибридный. Только статический подход дает мало информации, поэтому используется, в основном, динамический и гибридный. Это означает, что сбор признаков (характеристик) должен быть автоматизирован. Модели машинного обучения привнесли вместе с собой и новый класс кибератак – состязательные атаки. Модели машинного обучения, используемые для поиска вредоносного программного обеспечения (включая и большие языковые модели) не являются исключением и также подвержены такого рода атакам.**

***Ключевые слова*— машинное обучение, глубокое обучение, вредоносное программное обеспечение.**

I. Введение

Как обычно для многих (практически всех) современных работ, под системами искусственного интеллекта понимаются модели машинного (глубокого) обучения. Общие вопросы использования систем искусственного интеллекта в кибербезопасности были рассмотрены в нашей работе [1]. Все применения можно разделить на 4 группы:

- Искусственный интеллект в киберзащите
- Искусственный интеллект в нападении
- Безопасность самих систем искусственного интеллекта (атаки на модели машинного обучения и защита от них)
- Злонамеренные воздействия (разного рода фейки)

Данная статья посвящена киберзащите с использованием искусственного интеллекта, а более конкретно – поиску вредоносного программного обеспечения.

Статья получена 10 марта 2024.
Д.Е. Намиот – МГУ имени М.В. Ломоносова (email: dnamiot@gmail.com)
Е.А. Ильюшин – МГУ имени М.В. Ломоносова (email: john.ilyushin@gmail.com)

Вредоносное программное обеспечение (ВПО) – это совокупность вредоносных программ, состоящая из исполняемого кода, который может проникнуть в систему безопасности организации.
Обычно все потенциально нежелательное программное обеспечение, такое как вирусы, трояны, черви, рекламное программное обеспечение (ПО), программы-вымогатели, шпионские программы, кейлоггеры, руткиты и т. д., называют вредоносным ПО.

В 2023 году в мире было зарегистрировано более 6 млрд. атак, связанных с вредоносным программным обеспечением [2]. Согласно доступному отчету PurpleSec Trend Report [3], во время пандемии был зафиксирован рост на 600% в запуске кибератак с помощью вредоносного ПО. Количество кибератак постоянно растет, и они стали реальным разрушительным явлением.

Кроме того, в последнее время, наблюдается рост числа атак вредоносного ПО, нацеленных на различные отрасли, включая государственные сервисы, образование, финансовую отрасль, здравоохранение и т.п. Соответственно, поиск (нейтрализация) вредоносного программного обеспечения – это классическая и очень важная задача кибербезопасности.

Вопросы использования систем машинного (глубокого) обучения для киберзащиты рассматриваются, по меньшей мере, в двух магистратурах факультета ВМК МГУ имени М.В. Ломоносова. Это магистерская программа Искусственный интеллект в кибербезопасности [4] и магистерская программа Кибербезопасность [5].

Оставшаяся часть статьи структурирована следующим образом. В разделе II рассматриваются общие вопросы использования машинного обучения для поиска вредоносного программного обеспечения. В разделе III мы рассматриваем примеры систем, использующих машинное (глубокое) обучение для поиска вредоносного программного обеспечения. В разделе IV мы останавливаемся на возможном использовании больших языковых моделей (LLM) для такого класса задач.

II. ВПО и машинное обучение

Обычно вредоносное ПО классифицируется в зависимости от его функциональности, способа

распространения и воздействия на систему. Безопасность взаимосвязанных устройств имеет первостепенное значение для современных организаций, поскольку киберугрозы продолжают развиваться и представляют собой значительные риски. Понимание различных типов вредоносных программ и механизмов их заражения имеет решающее значение для того, чтобы организации могли эффективно защитить свои системы и данные.

Анализ вредоносного ПО включает в себя методы или инструменты для понимания его поведения с целью построения стратегий защиты и подготовки систем к противостоянию будущим атакам. Поставщики средств защиты от вредоносных программ обычно используют сигнатурные и эвристические подходы для обнаружения и удаления вредоносных файлов, прежде чем они нанесут ущерб системе. Первый метод сопоставляет строки, извлеченные из подозрительных образцов, с известным хранилищем сигнатур. Этот метод не подходит для обнаружения новых штаммов вредоносного ПО. Кроме того, управлять огромной базой подписей практически невозможно.

Как правило, эксперты подразделяют эвристические подходы на статический, динамический и гибридный анализ. Статический анализ проверяет исходный код путем декомпиляции предположительно вредоносного программного обеспечения. Этот метод не позволяет идентифицировать зашифрованный и запутанный код, поскольку он препятствует обратному проектированию. И наоборот, динамические подходы выполняют анализ путем распаковки и выполнения подозрительного двоичного кода на виртуальной машине или в песочнице. Однако такие методы имеют ограниченное покрытие кода и требуют больших вычислительных затрат.

Модели машинного (глубокого) обучения в данной области делают ровно то, что и в других областях [6]. Мы исходим из того, есть набор измерений (метрик), который характеризует статические характеристики и/или поведенческие аспекты исследуемого программного обеспечения. Предложить какую-либо аналитическую модель, которая позволила бы вычислять ВПО, практически никогда не удается. Управлять большой базой эвристических правил (продукций) реально очень трудно. В таких случаях мы и обращаемся к моделям машинного обучения с надеждой на то, что наша модель изучит скрытые закономерности и научится принимать решение. При этом, как и везде, мы ожидаем, что обученная модель выйдет за рамки тренировочных данных, научится обобщать (генерализация в моделях [7]) и будет принимать корректные решения на данных, которые отсутствовали в тренировочном наборе. При этом нам также важно понимание (основания) причин принятия решений моделью (объяснимые модели [8]), мы ожидаем, что модель будет обладать устойчивостью и надежностью [9]. То есть, никаких принципиальных отличий для схемы использования машинного обучения в данной предметной области от других областей нет. Основной момент, на котором делается акцент в следующем разделе – это данные, которые используются для анализа. Сбор данных и то, что называется feature engineering – это и есть главный момент (как, впрочем, и для других прикладных задач машинного обучения). Для LLM, использование которых рассматривается в IV разделе, ситуация не меняется – для осуществления анализа (классификации) необходимо сформировать контекст (промпт).

При этом необходимо понимать, что модели машинного обучения, используемые в киберзащите, становятся объектами состязательных атак. Причем такие атаки существуют на всех стадиях конвейера машинного обучения. Например, в работе [10] описываются атаки отравления, направленные, как раз, против моделей определения ВПО.

III. Примеры систем определения ВПО

В настоящем разделе мы останавливаемся на конкретных системах. Упор, как говорилось выше, будет сделан на анализе данных.

В работе [11] представлена интересная модель определения ВПО на платформе Android. Это DL-Droid фрейворк для определения ВПО. Платформа DL-Droid основана на динамическом подходе к генерации входных данных. На реальных устройствах запускались реальные приложения, количеством 31125, в процессе работы которых регистрировались 420 статических и динамических характеристик. Из этих приложений 11505 были ВПО, остальные (проверялось сторонним сервисом от McAffe) – чистые.

Авторы отмечают, что динамическое извлечение данных из реальных устройств также позволяет системе преодолеть ограничения статического анализа, например. динамическая загрузка кода, обфускация, шифрование данных и т. д.

Такой процесс нельзя поддержать вручную, поэтому использовались средства автоматизации. Система DynaLog [12] предназначена для автоматического запуска большого количества приложений Android с использованием либо эмулятора (виртуального устройства Android «AVD»), либо реального телефона, а также регистрации и извлечения нескольких динамических функций (например, вызовов API, действий/события приложения). При динамическом анализе приложений Android необходима генерация тестовых входных данных, чтобы обеспечить достаточное покрытие кода для запуска вредоносного поведения. DynaLog способен использовать различные методы генерации тестовых входных данных, что выполняется с помощью других инструментальных средств (Monkey [13], DroidBot [14] и др.).

The Monkey — это программа, которая запускается на эмуляторе или устройстве и генерирует псевдослучайные потоки пользовательских событий, таких как клики, прикосновения к экрану, вводы символов и т.д., а также ряд событий системного уровня. Вы можете использовать Monkey для стресс-тестирования разрабатываемых вами приложений

произвольным, но повторяемым образом. Инструмент Monkey генерирует псевдослучайные потоки событий благодаря генератору псевдослучайных чисел, управляемому начальным числом. Псевдослучайный поток событий по-прежнему является случайным подходом, поскольку выбор событий не основан на заранее определенном шаблоне, хотя его можно настроить. Важно отметить, что “случайный” здесь относится к выбору следующего события для выполнения, т. е. не соблюдается какой-либо конкретный шаблон, в отличие от подхода с отслеживанием состояния, где событие для выполнения выбирается на основе оценки текущего состояния (т. е. состояния пользовательского интерфейса). Monkey основан на подходе без сохранения состояния, а DroidBot – на подходе с сохранением состояния.

Использовались телефоны восьми разных марок. Каждый смартфон обрабатывал в среднем 100 приложений ежедневно. На SD-картах содержалась папка, полная различных ресурсов, таких как изображения, текстовые файлы, видео, звуковые файлы и т. д., имитирующих обычный телефон. Кроме того, каждый телефон был оснащен SIM-картой с балансом звонков, позволяющим использовать данные 3G, отправлять текстовые сообщения и даже совершать телефонные звонки по запросу. Телефоны также были подключены к Wi-Fi, чтобы тестируемые приложения могли, при необходимости, подключаться к внешним серверам.

То есть, по сути, начальный сбор данных – это автоматизированный фаззинг с регистрацией результатов.

Время выполнения было разным при каждом запуске и определялось выбранной генерацией тестовых входных данных. Было определено экспериментально, сколько времени необходимо для запуска каждого возможного события с использованием либо инструмента с отслеживанием состояния (DroidBot), либо инструмента без отслеживания состояния (Monkey). Для метода с отслеживанием состояния 180 сек. оказалось достаточным. Для генерации без сохранения состояния с использованием Monkey 300 сек. было достаточно, чтобы сгенерировать 4000 событий для приложений. После 4000 событий, большинство приложений не генерировали дальнейший динамический вывод от Dynalog.

Далее используется алгоритм InfoGain [15] для ранжирования признаков. InfoGain исследует изменение энтропии от добавления признака. Понятный по своему назначению алгоритм, который позволяет оценить вклад признака в принятие решения. В итоге получается ранжированный список признаков. Он зависит от того, каким образом задавались исходные данные (с учетом или без учета состояния, как было описано выше). На рисунке 1 представлен фрагмент такого списка.

Feature	Malware	Bening	InfoGain score
TelephonyManager;->getDeviceId	4899	1078	0.099
com.android.vending.INSTALL_REFERRER	741	3890	0.096
action.SMS_RECEIVED	2421	375	0.096
TelephonyManager;->getSubscriberId	1993	203	0.057
action.USER_PRESENT	2633	545	0.056
methods/HttpPost;-><init>	3408	1044	0.054
TelephonyManager;->getLine1Number	1429	163	0.043
WifiManager;->getConnectionInfo	2680	792	0.039
content/Context;->bindService	573	2271	0.037
Ljava/util/TimerTask;-><init>	7399	4068	0.033

Рис. 1. Отбор признаков [11]

Показанные в первом столбце признаки включают события и программные вызовы. Заметно, например, что вредоносные приложения гораздо чаще запрашивали ID устройства (4899 раз против 1078).

На основании рейтинга IngoGain были отобраны 120 характеристик, и к ним добавлены 300 разрешений (запрашиваемые приложениями разрешения по доступу к различным данным). Распределение найденных признаков представлено на рисунке 2

Feature set	No. of features
Application attributes features	97
Actions/Events features	23
Permission features	300
Total	420

Рис. 2. Найденные признаки [11].

После этого, собственно говоря, и начиналось машинное обучение. Весь этот процесс проиллюстрирован на рисунке 3.

Рассматривается задача двоичной классификации (Malware, Bening) на размеченном наборе данных. Непосредственно в данной работе авторы начинают с многослойного перцептрона [16]. С перцептроном проводятся эксперименты по подбору количества слоев и нейронов. Далее полученные метрики качества модели сравниваются с метриками от других алгоритмов Support Vector Machine (SVM Linear), Support Vector Machine with radial basis function kernel (SVM RBF), Naive Bayes (NB), Simple Logistic (SL), Partial Decision Trees (PART), Random Forest (RF), and J48 Decision Tree (C 4.5). Для сравнения используются стандартные метрики: true positive ratio (TPR), true negative ratio (TNR), false positive ratio (FPR), false negative ratio (FNR), и Precision. Изначально никаких предположений о том, какой алгоритм может быть более предпочтительным, нет, поэтому, по факту, проводится просто перебор методов для нахождения наилучших показателей.

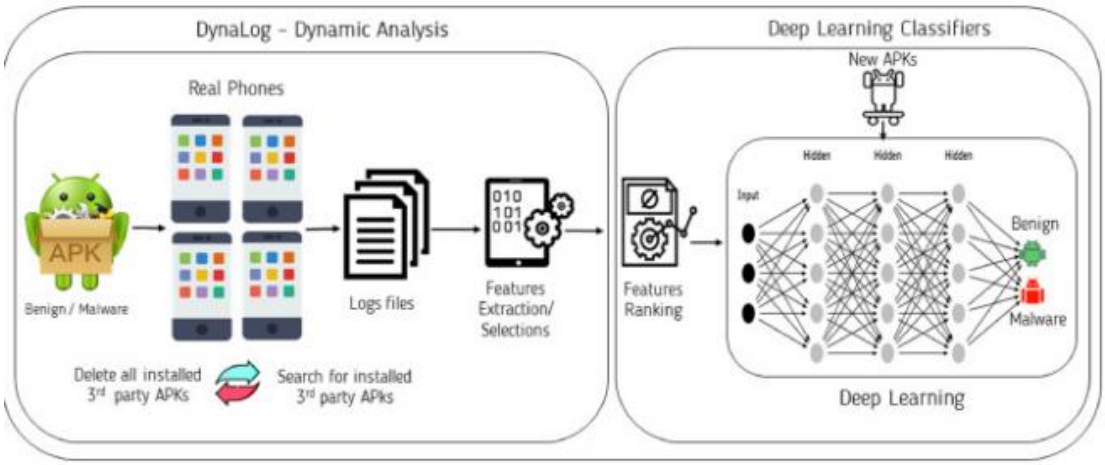

Рис.3. Сбор данных и тренировка классификатора [11]

На рисунке 4 приводится фрагмент таблицы результатов

	TPR	TNR	FPR	FNR	Precision	Recall
NB	0.816	0.886	0.114	0.184	0.86	0.86
SL	0.871	0.957	0.043	0.129	0.925	0.925
SVM RBF	0.871	0.957	0.043	0.129	0.925	0.925
SVM Linear	0.875	0.964	0.036	0.125	0.931	0.931
J48	0.919	0.967	0.033	0.081	0.949	0.949
PART	0.931	0.968	0.032	0.069	0.954	0.954
RF	0.941	0.988	0.012	0.059	0.971	0.971

Рис.4. Полученные метрики для разных моделей машинного обучения [11].

Что можно сказать по данному решению? Очевидно, часть, касающаяся собственно машинного обучения, является абсолютно технической, реализуемой множественными способами с помощью готовых фреймворков. Основная проблема – это сбор и подготовка данных. Второй момент, который необходимо отметить – это то, что никаких специальных действий по защите от возможных состязательных атак не предусматривается. Совершенно неудивительно, что состязательные атаки на модели защиты с помощью машинного обучения весьма распространены. Примеры можно найти в работах [17,18]. Специфика в осуществлении таких атак, безусловно, есть. Например, для атак уклонения мы, очевидно, не можем произвольным образом модифицировать входные данные, которые должны оставаться корректными бинарными файлами. Для рассмотренного подхода и сам датасет и информация о ранжировании признаков должны оставаться закрытыми, поскольку они позволят атакующему построить теневую модель и тестировать ВПО, которое будет проходить через фильтр-определитель [19]. Вопрос с датасетом является особенно острым. Поскольку сбор данных, как описано выше, является довольно трудоемким, то существует большая вероятность того, что разработчики захотят воспользоваться готовыми датасетами. Датасет может быть публичным, но вот информация об использовании его в конкретном продукте публичной не является. В противном случае мы будем иметь классический пример из атласа MITRE [20]. Открытая разведка (OSINT) получает информацию о тренировочном наборе, на нем атакующий строит теневую модель и отрабатывает атаку белого ящика.

В обзорной работе [21] приводится достаточно большой список подходов к определению ВПО с помощью глубокого обучения. Помимо множества примеров, где, подобно разобранному выше случаю, собирают общий набор характеристик, можно выделить подходы, связанные с анализом поведения и графом потоков данных [22, 23].

IV. LLM в определении вредоносного программного обеспечения

Недавние достижения в разработке больших языковых моделей (LLM) для обработки естественного языка (NLP), привели к появлению новых многообещающих методологий для усиления мер кибербезопасности [24]. Некоторые сценарии приложений включают, среди прочего, фильтрацию спама, аудит смарт-контрактов или анализ кода. Среди этих решений трансферное обучение оказывается особенно эффективной стратегией. Этот подход использует обширную базу знаний LLM, первоначально обученных на больших наборах данных, для адаптации их к специализированным областям с относительно скудными данными, такими как обнаружение вредоносного ПО [25].

Примером такой работы является фреймворк, описанный в работе [26]. Здесь предлагается дообучить LLM размеченными данными о потоках системных вызовов во вредоносных и “чистых” программах. Авторы исходят из того, что такие цепочки вызовов используются для определения ВПО (в том числе, и средствами машинного обучения, так, как это было описано в разделе III). Типичный подход – это использование n-gram [27]. Последовательность системных вызовов рассматривается как текст (заметим, что главная проблема – это снова сбор данных) и признаки (features) – это последовательные цепочки символов фиксированной длины. Данные получаются простыми и понятными. Но, как отмечается в [26], их основное ограничение заключается в недостаточном понимании контекста, поскольку n-граммы рассматривают только последовательности фиксированной длины и не могут фиксировать более

длинные зависимости или сложное поведение, выходящее за пределы определенного окна размера n. Это часто приводит к более высокому уровню ложноположительных и отрицательных результатов, особенно в случае сложных вредоносных программ, которые используют методы уклонения или имитируют безопасное поведение.

В работе [28] сравниваются результаты работы моделей с LSTM, Transformers и Longformers (вариант Transformer с меньшей сложностью) с 4-граммовыми статистическими подходами для классификации системных вызовов. Они опубликовали набор данных, охватывающий системные вызовы, генерируемые семью поведениями, отслеживаемыми в двух миллионах веб-запросов. Решения на основе нейронных сетей показали более высокую производительность, особенно для LSTM и Transformer. Вместе с тем отмечены проблемы использования сложных языковых моделей в реальном времени.

Что конкретно делают авторы в работе [26]? Во-первых, собирают исходные данные для обучения. 1Tb датасет, содержащий цепочки системных вызовов “вредоносных” и “чистых” программ. Данные собирались в формате *[timestamp, process, PID, syscall]*. Инструмент для сбора данных: perf trace. На этапе предобработки данные упрощались до временной метки и системного вызова без параметров. В качестве ВПО выступали известные приложения:

- Bashlite [29] — известное семейство ботнетов, нацеленное на устройства IoT. Способно запускать распределенные атаки типа «отказ в обслуживании» (DDoS), выполнять произвольные команды оболочки и включать зараженные устройства в ботнет.
- TheTick [30], бэкдор, используемый для удаленного управления ботами через сервер, использующий удаленную оболочку и получающий данные с целевых устройств.
- Bdvl [31] — руткит с очень широким функционалом. Он варьируется от скрытых бэкдоров, которые позволяют использовать несколько методов подключения, до кейлогинга и кражи паролей и файлов.
- RansomwarePoC [32] — пример реализации программы-вымогателя на Python с полными возможностями шифрования.

Цепочки вызовов помечались одним из 5 классов (“чистое ПО” и 4 класса для ВПО). К выбранным для проверки концепции Open Source моделям добавлялся последний уровень из 5 нейронов (один нейрон на класс) для задачи классификации. Далее – тренировка на 5 эпох. Общая схема процесса представлена на рисунке 5.

Анализ других работ по указанной тематике показывает ту же самую схему – дообучение LLM для новых задач [33].

Что можно отметить относительно данного подхода? Классификатор работает против тех ВПО, которые были в тренировочном наборе. Вопрос с генерализацией подобного подхода остается открытым. Сможет ли дообученная LLM определить новый класс ВПО, отнеся его к одному из 4 ей известных? Также можно отметить, что состязательные атаки работают и против LLM. В данном случае знание подхода (цепочка системных вызовов) и списка использованных ВПО может помочь атакующему построить состязательный пример.

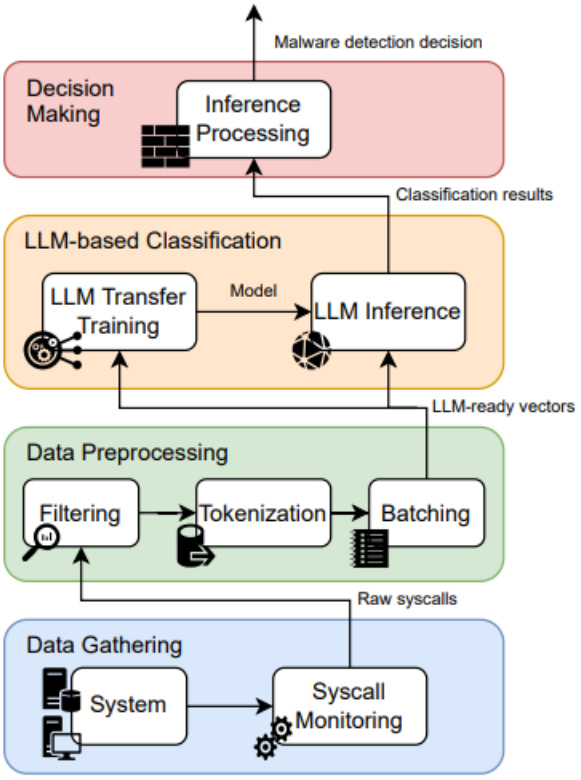

Рис. 5. Определение ВПО с помощью LLM [26]

Схема проста для понимания и работает, но вот метрики в таком подходе были не самые высокие (рис. 6)

Model	Context Size	Accuracy	Precision	Recall	F1-Score	Kappa	MCC
BERT	512	0.6772	0.8200	0.6596	0.6465	0.5504	0.6024
DistilBERT	512	0.6289	0.7100	0.6181	0.5930	0.4786	0.5379
GPT-2	1024	0.6944	0.7986	0.6865	0.6808	0.5792	0.6123
BigBird	4096	0.8667	0.8754	0.8668	0.8688	0.8298	0.8311
Longformer	4096	0.8616	0.8696	0.8614	0.8621	0.8232	0.8250
Mistral	8192	0.5817	0.6112	0.6462	0.6242	0.4754	0.4798

Рис. 6. Полученные метрики [26].

V. Заключение

В настоящей статье мы попытались подробно расписать техническую сторону использования машинного обучения при поиске вредоносного программного обеспечения. Основная проблема в подобного рода системах – это сбор и подготовка данных. Часть работы, связанная непосредственно с машинным (глубоким) обучением является чисто технической. Также обращает на себя внимание отсутствие противодействия возможным состязательным атакам.

Благодарности

Авторы благодарны сотрудникам лаборатории Открытых информационных технологий кафедры Информационной безопасности факультета ВМК МГУ имени М.В. Ломоносова за обсуждения и ценные замечания.

Статья продолжает серию публикаций, начатых работами об использовании искусственного интеллекта в кибербезопасности [34]. Традиционно, отметим, что все публикации в журнале INJOIT, связанные с цифровой повесткой, начинались с работ В.П. Куприяновского и его соавторов [35-37].

Библиография

[1] Namiot, Dmitry, Eugene Ilyushin, and Ivan Chizhov. "Artificial intelligence and cybersecurity." International Journal of Open Information Technologies 10.9 (2022): 135-147. (in Russian).

[2] Annual number of malware attacks worldwide from 2015 to 2023 https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/ Retrieved: Mar, 2024

[3] Cyber Security Trends Report https://purplesec.us/resources/cyber-security-statistics/ Retrieved: 2024.

[4] Магистерская программа «Искусственный интеллект в кибербезопасности» (ФГОС) https://cs.msu.ru/node/3732 Retrieved: Mar, 2024

[5] Магистерская программа «Кибербезопасность» https://cyber.cs.msu.ru/ Retrieved: Mar, 2024

[6] Asmitha, K. A., et al. "Deep learning vs. adversarial noise: a battle in malware image analysis." Cluster Computing (2024): 1-30.

[7] Neyshabur, Behnam, et al. "Exploring generalization in deep learning." Advances in neural information processing systems 30 (2017).

[8] Zhang, Zhibo, et al. "Explainable artificial intelligence applications in cyber security: State-of-the-art in research." IEEE Access 10 (2022): 93104-93139.

[9] Robey, Alexander, Hamed Hassani, and George J. Pappas. "Model-based robust deep learning: Generalizing to natural, out-of-distribution data." arXiv preprint arXiv:2005.10247 (2020)..

[10] Narisada, Shintaro, et al. "Stronger targeted poisoning attacks against malware detection." International Conference on Cryptology and Network Security. Cham: Springer International Publishing, 2020.

[11] Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "DL-Droid: Deep learning based android malware detection using real devices." Computers & Security 89 (2020): 101663.

[12] Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "DynaLog: An automated dynamic analysis framework for characterizing android applications." 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). IEEE, 2016.

[13] Monkey https://developer.android.com/studio/test/other-testing-tools/monkey Retrieved: Mar, 2024

[14] Li, Yuanchun, et al. "Droidbot: a lightweight ui-guided test input generator for android." 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, 2017.

[15] Pereira, Rafael Barros, et al. "Information gain feature selection for multi-label classification." (2015).

[16] Amato, Flora, et al. "Multilayer perceptron: an intelligent model for classification and intrusion detection." 2017 31st International conference on advanced information networking and applications workshops (WAINA). IEEE, 2017.

[17] Suciu, Octavian, Scott E. Coull, and Jeffrey Johns. "Exploring adversarial examples in malware detection." 2019 IEEE Security and Privacy Workshops (SPW). IEEE, 2019.

[18] Kolosnjaji, Bojan, et al. "Adversarial malware binaries: Evading deep learning for malware detection in executables." 2018 26th European signal processing conference (EUSIPCO). IEEE, 2018.

[19] Namiot, Dmitry. "Schemes of attacks on machine learning models." International Journal of Open Information Technologies 11.5 (2023): 68-86.

[20] MITRE ATLAS https://atlas.mitre.org/ Retrieved: Apr 2024

[21] Tayyab, Umm-e-Hani, et al. "A survey of the recent trends in deep learning based malware detection." Journal of Cybersecurity and Privacy 2.4 (2022): 800-829.

[22] Chen, Yun-Chun, et al. "Deep learning for malicious flow detection." 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC). IEEE, 2017.

[23] Zhu, Dali, et al. "DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data." 2017 IEEE symposium on computers and communications (ISCC). IEEE, 2017.

[24] Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, et. al. Large language models in cybersecurity: State-of-the-art. arXiv preprint arXiv:2402.00891, 2024.

[25] Andrei Kucharavy, Zachary Schillaci, Lo¨ıc Marechal, et. al. Fundamentals of generative large language models and perspectives in cyber-defense. arXiv preprint arXiv:2303.12132, 2023.

[26] Sánchez, Pedro Miguel Sánchez, et al. "Transfer Learning in Pre-Trained Large Language Models for Malware Detection Based on System Calls." arXiv preprint arXiv:2405.09318 (2024).

[27] Muhammad Ali, Stavros Shiaeles, Gueltoum Bendiab et.al. Machine learning and n-gram malware feature extraction and detection system. Electronics, 9(11):1777, 2020.

[28] Quentin Fournier, Daniel Aloise, and Leandro R Costa. Language models for novelty detection in system call traces. arXiv preprint arXiv:2309.02206, 2023.

[29] Hammerzeit. BASHLITE. https://github.com/hammerzeit/BASHLITE, Retrieved: Apr, 2024.

[30] Nccgroup. The Tick – A simple embedded Linux backdoor. https://github.com/nccgroup/thetick/, Retrieved: Apr, 2024

[31] Error996. bedevil (bdvl). https://github.com/Error996/bdvl/, Retrieved: Apr, 2024

[32] Jimmyly00. Ransomware PoC GitHub repository. https://github.com/jimmy-ly00/Ransomware-PoC, Retrieved: Apr, 2024

[33] LLM for malware detection. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=llm+for+malware+detection Retrieved: Mar, 2024

[34] Ilyushin, Eugene, Dmitry Namiot, and Ivan Chizhov. "Attacks on machine learning systems-common problems and methods." International Journal of Open Information Technologies 10.3 (2022): 17-22.

[35] Цифровая экономика и Интернет Вещей - преодоление силоса данных / В. П. Куприяновский, А. Р. Ишмуратов, Д. Е. Намиот [и др.] // International Journal of Open Information Technologies. – 2016. – Т. 4, № 8. – С. 36-42. – EDN WFVAPB.

[36] Искусственный интеллект как стратегический инструмент экономического развития страны и совершенствования ее государственного управления. Часть 2. Перспективы применения искусственного интеллекта в России для государственного управления / И. А. Соколов, В. И. Дрожжинов, А. Н. Райков [и др.] // International Journal of Open Information Technologies. – 2017. – Т. 5, № 9. – С. 76-101. – EDN ZEQDMT.

[37] Розничная торговля в цифровой экономике / В. П. Куприяновский, С. А. Синягов, Д. Е. Намиот [и др.] // International Journal of Open Information Technologies. – 2016. – Т. 4, № 7. – С. 1-12. – EDN WCMIWN.

Artificial Intelligence in Cybersecurity: Finding Malware

Dmitry Namiot, Eugene Ilyushin

***Abstract* - In this article, we examine in detail the technical aspects of searching (classifying) malicious software using machine (deep) learning. We consider both a classic example of classification using traditional models such as a multilayer perceptron, SVM, Random Forest, etc., and approaches associated with the use of large language models. The main problem in this type of task lies in the area of data preparation and collection. Code analysis for malware can be static, dynamic (runtime), or hybrid. Only the static approach provides little information, so mainly dynamic and hybrid ones are used. This means that the collection of features (characteristics) must be automated. Machine learning models have brought with them a new class of cyber attacks: adversarial attacks. Machine learning models used to find malware (including large language models) are no exception and are also susceptible to these types of attacks.**

***Keywords* - machine learning, deep learning, malware.**

REFERENCES

[1] Namiot, Dmitry, Eugene Ilyushin, and Ivan Chizhov. "Artificial intelligence and cybersecurity." International Journal of Open Information Technologies 10.9 (2022): 135-147. (in Russian).

[2] Annual number of malware attacks worldwide from 2015 to 2023 https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/ Retrieved: Mar, 2024

[3] Cyber Security Trends Report https://purplesec.us/resources/cyber-security-statistics/ Retrieved: 2024.

[4] Magisterskaja programma «Iskusstvennyj intellekt v kiberbezopasnosti» (FGOS) https://cs.msu.ru/node/3732 Retrieved: Mar, 2024

[5] Magisterskaja programma «Kiberbezopasnost'» https://cyber.cs.msu.ru/ Retrieved: Mar, 2024

[6] Asmitha, K. A., et al. "Deep learning vs. adversarial noise: a battle in malware image analysis." Cluster Computing (2024): 1-30.

[7] Neyshabur, Behnam, et al. "Exploring generalization in deep learning." Advances in neural information processing systems 30 (2017).

[8] Zhang, Zhibo, et al. "Explainable artificial intelligence applications in cyber security: State-of-the-art in research." IEEE Access 10 (2022): 93104-93139.

[9] Robey, Alexander, Hamed Hassani, and George J. Pappas. "Model-based robust deep learning: Generalizing to natural, out-of-distribution data." arXiv preprint arXiv:2005.10247 (2020)..

[10] Narisada, Shintaro, et al. "Stronger targeted poisoning attacks against malware detection." International Conference on Cryptology and Network Security. Cham: Springer International Publishing, 2020.

[11] Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "DL-Droid: Deep learning based android malware detection using real devices." Computers & Security 89 (2020): 101663.

[12] Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "DynaLog: An automated dynamic analysis framework for characterizing android applications." 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). IEEE, 2016.

[13] Monkey https://developer.android.com/studio/test/other-testing-tools/monkey Retrieved: Mar, 2024

[14] Li, Yuanchun, et al. "Droidbot: a lightweight ui-guided test input generator for android." 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, 2017.

[15] Pereira, Rafael Barros, et al. "Information gain feature selection for multi-label classification." (2015).

[16] Amato, Flora, et al. "Multilayer perceptron: an intelligent model for classification and intrusion detection." 2017 31st International conference on advanced information networking and applications workshops (WAINA). IEEE, 2017.

[17] Suciu, Octavian, Scott E. Coull, and Jeffrey Johns. "Exploring adversarial examples in malware detection." 2019 IEEE Security and Privacy Workshops (SPW). IEEE, 2019.

[18] Kolosnjaji, Bojan, et al. "Adversarial malware binaries: Evading deep learning for malware detection in executables." 2018 26th European signal processing conference (EUSIPCO). IEEE, 2018.

[19] Namiot, Dmitry. "Schemes of attacks on machine learning models." International Journal of Open Information Technologies 11.5 (2023): 68-86.

[20] MITRE ATLAS https://atlas.mitre.org/ Retrieved: Apr 2024

[21] Tayyab, Umm-e-Hani, et al. "A survey of the recent trends in deep learning based malware detection." Journal of Cybersecurity and Privacy 2.4 (2022): 800-829.

[22] Chen, Yun-Chun, et al. "Deep learning for malicious flow detection." 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC). IEEE, 2017.

[23] Zhu, Dali, et al. "DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data." 2017 IEEE symposium on computers and communications (ISCC). IEEE, 2017.

[24] Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, et. al. Large language models in cybersecurity: State-of-the-art. arXiv preprint arXiv:2402.00891, 2024.

[25] Andrei Kucharavy, Zachary Schillaci, Lo¨ıc Marechal, et. al. Fundamentals of generative large language models and perspectives in cyber-defense. arXiv preprint arXiv:2303.12132, 2023.

[26] Sánchez, Pedro Miguel Sánchez, et al. "Transfer Learning in Pre-Trained Large Language Models for Malware Detection Based on System Calls." arXiv preprint arXiv:2405.09318 (2024).

[27] Muhammad Ali, Stavros Shiaeles, Gueltoum Bendiab et.al. Machine learning and n-gram malware feature extraction and detection system. Electronics, 9(11):1777, 2020.

[28] Quentin Fournier, Daniel Aloise, and Leandro R Costa. Language models for novelty detection in system call traces. arXiv preprint arXiv:2309.02206, 2023.

[29] Hammerzeit. BASHLITE. https://github.com/hammerzeit/BASHLITE, Retrieved: Apr, 2024.

[30] Nccgroup. The Tick – A simple embedded Linux backdoor. https://github.com/nccgroup/thetick/, Retrieved: Apr, 2024

[31] Error996. bedevil (bdvl). https://github.com/Error996/bdvl/, Retrieved: Apr, 2024

[32] Jimmyly00. Ransomware PoC GitHub repository. https://github.com/jimmy-ly00/Ransomware-PoC, Retrieved: Apr, 2024

[33] LLM for malware detection. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=llm+for+malware+detection Retrieved: Mar, 2024

[34] Ilyushin, Eugene, Dmitry Namiot, and Ivan Chizhov. "Attacks on machine learning systems-common problems and methods." International Journal of Open Information Technologies 10.3 (2022): 17-22.

[35] Cifrovaja jekonomika i Internet Veshhej - preodolenie silosa dannyh / V. P. Kuprijanovskij, A. R. Ishmuratov, D. E. Namiot [i dr.] // International Journal of Open Information Technologies. – 2016. – T. 4, # 8. – S. 36-42. – EDN WFVAPB.

[36] Iskusstvennyj intellekt kak strategicheskij instrument jekonomicheskogo razvitija strany i sovershenstvovanija ee gosudarstvennogo upravlenija. Chast' 2. Perspektivy primenenija iskusstvennogo intellekta v Rossii dlja gosudarstvennogo upravlenija / I. A. Sokolov, V. I. Drozhzhinov, A. N. Rajkov [i dr.] // International Journal of Open Information Technologies. – 2017. – T. 5, # 9. – S. 76-101. – EDN ZEQDMT.

[37] Roznichnaja torgovlja v cifrovoj jekonomike / V. P. Kuprijanovskij, S. A. Sinjagov, D. E. Namiot [i dr.] // International Journal of Open Information Technologies. – 2016. – T. 4, # 7. – S. 1-12. – EDN WCMIWN.