© 2024 г. Т.М. БИДЖИЕВ (temirlanbid@gmail.com),
Д.Е. НАМИОТ, д-р техн. наук (dnamiot@gmail.com)
(Московский государственный университет им. М.В. Ломоносова)

АТАКИ НА МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ, ОСНОВАННЫЕ НА ФРЕЙМВОРКЕ PYTORCH

Рассматриваются последствия использования облачных сервисов для обучения нейронных сетей с точки зрения кибербезопасности. Ресурсоемкость обучения нейронных сетей создает проблемы, что приводит к росту зависимости от облачных сервисов. Однако такая зависимость создает новые риски кибербезопасности. Исследование посвящено новому методу атаки, использующему веса нейронных сетей для незаметного распространения скрытых вредоносных программ. Рассматриваются семь методов встраивания и четыре типа триггеров для активации вредоносного программного обеспечения. Представлен фреймворк с открытым исходным кодом, автоматизирующий внедрение кода в весовые параметры нейронных сетей, что позволяет исследователям изучать и противодействовать этому новому вектору атак.

Ключевые слова: нейронные сети, вредоносное программное обеспечение, стеганография, триггеры.

DOI: 10.31857/S0005231024030038, **EDN:** TZXTPW

1. Введение

Значительный прогресс и широкое применение машинного обучения в различных областях привели к возникновению важных вопросов, касающихся безопасности и надежности моделей машинного обучения [1, 2].

Одной из главных проблем безопасности в машинном обучении является уязвимость моделей к различного рода атакам [3, 4]. Новой тенденцией в современных системах машинного обучения является рост числа атак, направленных на внедрение вредоносного программного обеспечения (ПО) в нейронные сети [5–7]. Такая форма атаки представляет собой серьезную угрозу безопасности и надежности моделей, поскольку злоумышленники могут использовать их для выполнения вредоносных действий, обходя традиционные механизмы защиты. Такие атаки могут привести к скрытой активации вредоносных функций, утечке конфиденциальной информации или неправильной классификации данных, что подрывает точность моделей машинного обучения [8]. Поэтому понимание, обнаружение и защита от атак, связанных с внедрением вредоносных программ в нейронные сети, становятся важными задачами в области кибербезопасности.

Методы внедрения вредоносного ПО используются при поставке готовых моделей конечному пользователю через сервисы MLaaS (Machine Learning

as a Service). Часто потребители, не являющиеся экспертами в области машинного обучения, работают с сериями MLaaS, не имея представления об обучении, тестировании и обработке данных. Как правило, самым важным критерием для таких пользователей является точность модели. Злоумышленник может воспользоваться этим и незаметно для пользователя внедрить в модель глубокой нейронной сети вредоносное ПО.

Обзор методов внедрения вредоносных программ непосредственно в модели машинного обучения представлен в данной статье [9].

2. Постановка задачи

Целью данной работы является изучение и анализ атак на машинное обучение, направленных на внедрение вредоносного кода в нейронные сети, путем разработки специализированного фреймворка [10], автоматизирующего процесс внедрения вредоносного кода в веса нейронных сетей [11].

Этот фреймворк позволяет исследователям и специалистам проводить эксперименты и проверять устойчивость своих моделей машинного обучения, а также разрабатывать и применять контрмеры для защиты от подобных атак. Фреймворк обеспечивает гибкость и масштабируемость, позволяя настраивать и адаптировать методы внедрения вредоносного ПО в зависимости от конкретных требований и сценария использования.

Был проведен глубокий анализ существующих методологий и изучены подходы к реализации программного обеспечения. Данная работа вносит вклад в область безопасности машинного обучения и дает практическое представление о защите моделей машинного обучения от атак с внедрением кода на весовые коэффициенты моделей.

В следующих разделах рассмотрим теоретические основы, методологию исследования и экспериментальную оценку предлагаемого фреймворка, конечной целью которого является повышение безопасности и надежности систем машинного обучения.

3. Методология

На рис. 1 показан типичный сценарий взаимодействия между пользователем и злоумышленником. В этом сценарии пользователь, намеревающийся использовать нейросетевую модель в бизнес-целях, начинает процесс с выбора желаемой архитектуры модели. Затем пользователь приступает к обучению модели с помощью провайдеров MLaaS или получает предварительно обученную модель из различных источников.

В конкретном сценарии, где злоумышленник берет на себя роль сервиса MLaaS, предполагается, что он не обладает способностью изменять архитектуру полученной нейронной сети. Однако злоумышленник сохраняет возможность манипулировать весовыми параметрами сети. Это позволяет злоумышленнику вносить вредоносные модификации в модель, не изменяя ее фундаментальной структуры.

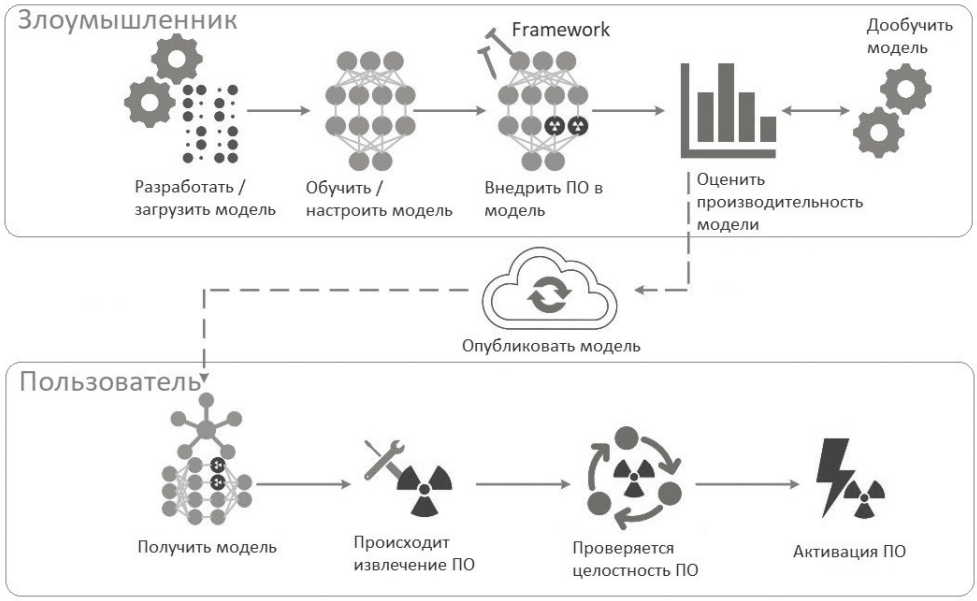

Рис. 1. Общий сценарий взаимодействия пользователя и злоумышленника.

Для того чтобы реализовать все цели, злоумышленнику необходимо выполнить действия в соответствии с рис. 1. В этом процессе злоумышленник начинает с получения модели нейронной сети, которая может быть предоставлена пользователем или разработана им самим. Затем он приступает к обучению модели до желаемого уровня точности, причем этот этап может быть выполнен как самостоятельно, так и передан на аутсорсинг провайдерам MLaaS. После успешного обучения злоумышленник подготавливает и внедряет в модель вредоносное ПО, одновременно отслеживая и контролируя потерю точности модели с помощью методов, подробно описанных в данной статье. Для развертывания и активации вредоносного ПО злоумышленник создает триггер, используя методы, рассмотренные в этом же исследовании. Когда модель полностью подготовлена, злоумышленник может использовать дополнительные техники, такие как “загрязнение цепочки” [12], чтобы распространить эту модель в публичных хранилищах или других местах.

Далее пользователь получает нейросеть со встроенным вредоносным ПО, которое будет извлечено и запущено при активации триггера.

4. Методы внедрения вредоносного ПО

4.1. Внедрение вредоносных байтов в нейроны

Согласно стандарту IEEE [13], число с плавающей точкой имеет размер 32 бита, где первый бит отводится под знак числа, следующие 8 бит – под экспоненту, а последние 23 бита – под мантиссу. В результате получается число $\pm 1.m \times 2^n$ в двоичной форме, где m – мантисса числа, n – экспонента.

Такое число принадлежит диапазону от 2^{-127} до $2^{127}-1$. Экспонента отвечает за его величину, т.е., сохранив несколько первых байт числа, оставшуюся часть можно заменить на вредоносные байты, сохранив небольшую разницу с исходным числом.

4.2. Методы StegoNet

В статье StegoNet [5] предлагается четыре метода внедрения вредоносного ПО. Давайте рассмотрим их.

LSB замена

Этот подход основан на использовании замены LSB (Least Significant Bits) [7], как это применяется в технике стеганографии [14]. Она включает в себя выбор количества нейронов и параметров, подходящих для вредоносной программы. Двоичный код вредоносной программы делится на сегменты длиной, равной выбранной длине замены LSB, и эти сегменты записываются в параметры модели вместо последних нескольких битов соответствующего параметра.

Это решение неприменимо к сильно сжатым моделям нейронных сетей. Например, размер модели MobileNet [15] составляет всего 4 МБ при 4 миллионах 8-битных параметров. Такие сжатые модели быстро теряют точность даже при незначительных изменениях параметров. Этот метод не подходит для сжатых моделей.

Устойчивое обучение

Удаление некоторого набора нейронов из топологии нейросетевой модели может привести к значительному снижению точности, но параметры, соединяющие оставшиеся нейроны, могут быть скорректированы (переобучены) для достижения первоначальной точности. На основе этой интуиции была предложена методика “Устойчивое обучение”.

Как показано на рис. 2,*а*, метод предполагает прямую замену всех битов выбранных параметров на сегменты вредоносных программ. Такие нейроны (т.е. с измененными параметрами) не будут обновляться в процессе переобучения. Ожидается, что после обучения точность модели будет восстановлена, что позволит скрыть наличие внедренного ПО.

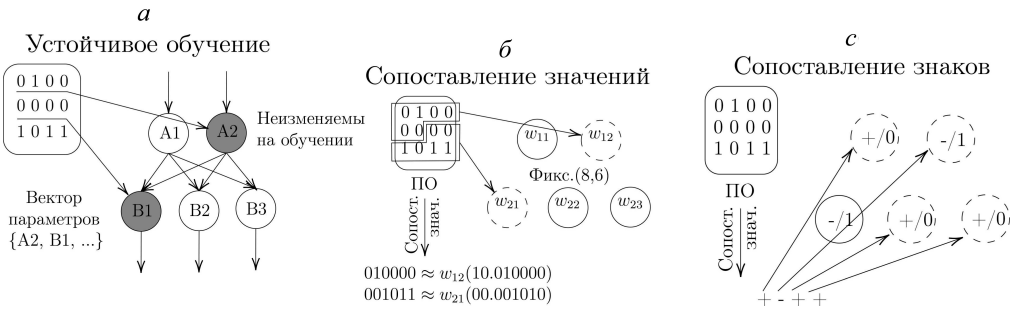

Рис. 2. Методы внедрения вредоносных программ [5].

Сопоставление значений

Предположим, есть модель с 8-битными числами с фиксированной точкой и шестью битами после десятичной точки. Чтобы облегчить сравнение значений, двоичный код вредоносной программы изначально делится на сегменты длиной, соответствующей количеству битов после десятичной точки. Затем для каждого сегмента выполняется полный исчерпывающий поиск параметров модели, чтобы найти (или заменить) такое же (или близкое) значение битов дробного параметра, см. пример на рис. 2,*б*. Наконец, сопоставляем сегмент кода с соответствующим параметром, при необходимости заменяя дробные биты параметра на сегмент кода.

Сопоставление знаков

Метод сопоставления знаков использует аналогичное правило "полного поиска и сопоставления", основанное на бите знака параметров модели. Как показано на рис. 2,*в*, метод сопоставления знаков проходит через параметры модели и сопоставляет бит знака параметра с каждым отдельным битом вредоносного кода. Например, 0 сопоставляется со знаком параметра +, таким образом в конечном итоге код сопоставляется с последовательностью битов знака для соответствующих параметров. Необходимо хранить вектор сопоставленных параметров.

4.3. Методы EvilModel

В статье EvilModel [6] предлагаются еще три метода. Рассмотрим их.

MSB сохранение

Поскольку наиболее важная экспоненциальная часть параметра находится в первом байте, первый байт является наиболее существенным для определения значения параметра. Поэтому его можно оставить без изменений и внедрить вредоносную программу в следующие три байта. Таким образом, значения параметров остаются в разумных пределах.

Быстрая замена

Если заменить параметры тремя байтами вредоносного кода и первым байтом префикса 0x3C или 0xBC в зависимости от знака параметра, большинство значений параметров все равно окажутся в разумных пределах. По сравнению с методом MSB, этот метод может оказать большее влияние на производительность модели, но поскольку ему не нужно разбивать параметры в нейроне, он будет работать быстрее.

Половинная замена

Как и в случае с методом MSB reservation, если оставить неизменными первые два байта, а не один, и изменить два других байта, значение этого числа будет колебаться в меньшем диапазоне. Однако, поскольку в параметре заменяются только два байта, этот метод может внедрить меньше данных, чем два предыдущих метода.

5. Сравнение методов внедрения

Давайте сравним точность различных моделей до и после применения всех описанных выше методов, см. [5, 6] и табл. 1.

Как видно из этой таблицы, наивный метод LSB замены может поддерживать хорошую точность тестирования на средних нейронных сетях, но это не относится к малым нейронным сетям. Например, он приводит к значительному снижению точности (т.е. резкому падению до $\approx 0{,}1\%$) в моделях нейронных сетей с высокой степенью сжатия из-за ограниченной точности данных и уменьшения количества параметров.

Напротив, метод устойчивого обучения может сравнительно лучше поддерживать вредоносные программы на небольших нейронных сетях. Для небольших вредоносных программ, таких как EquationDrug, ZeusVM и Cerber [16], он может поддерживать точность тестирования на уровне оригинала, даже в самых маленьких сетях MobileNet [15] (4,2 МБ) и SqueezeNet [17] (4,6 МБ). Однако точность MobileNet значительно снизилась с 66,7% до 0,7% при увеличении размера вредоносной программы с 0,59 МБ (Cerber) до 3,35 МБ (WannaCry). Можно заметить, что верхняя граница отношения размера вредоносной программы к размеру модели для метода Resilience training составляет $\approx$15% без снижения точности.

Такая проблема была устранена с помощью метода сопоставления значений, основанного на "полном поиске и отображении". Для сильно сжатых моделей нейронных сетей, таких как "Comp.Alexnet" [18], параметры модели сильно сжаты и метод может быть менее эффективным. Аналогичная тенденция прослеживается и в методе сопоставления знаков. В целом, однако, метод сопоставления знаков всегда может сохранить исходную точность тестирования для всех применимых случаев.

Для метода MSB reservation из-за избыточности нейросетевых моделей при внедрении вредоносного ПО точность тестирования не влияет на модели большого размера (>200 Мб). В некоторых случаях (например, Vgg 16 [19] с NSIS) точность немного увеличивается, что также отмечено в статье Stegonet [5]. При реализации вредоносного ПО с использованием MSB сохранения точность снижается по мере увеличения размера встроенного вредоносного ПО для моделей среднего и малого размера. Например, точность снижается на 5% для моделей среднего размера, таких как Resnet50 с Mamba. Теоретически максимальный порядок встраивания (т.е. отношение размера вредоносного кода к размеру модели) для метода MSB сохранения составляет 75%. В эксперименте верхняя граница порядка встраивания без существенного снижения точности составляет 25,73% (Googlenet с Artemis).

Эффективность метода быстрой замены схожа с методом MSB сохранения, но нестабильна для небольших моделей. Когда в модель среднего или малого размера внедряется более крупное вредоносное ПО, точность модели значительно снижается. Например, для Mobilenet с VikingHorde точность тестирования резко падает до 0,108%. Это показывает, что быстрая подстанов-

ка может быть использована вместо метода MSB сохранения, когда модель велика или задача требует много времени. В эксперименте порядок встраивания без существенного снижения точности составил 15,9% (Resnet18 с Viking Horde).

Метод половинной замены превосходит все остальные методы. Благодаря избыточности нейросетевых моделей точность после встраивания кода любого размера практически не снижается, даже если почти половина модели заменена вредоносными байтами, точность колеблется в пределах 0,01% от исходной. В Squeezenet небольшого размера (4,74 МБ) можно внедрить образец вируса Mamba размером 2,3 МБ, при этом точность увеличится на 0,048%. Теоретически максимальный порядок встраивания составляет 50%. В эксперименте было достигнуто близкое к теоретическому значение 48,52% (Squeezenet с Mamba).

6. Методы активации вредоносного ПО

6.1. Триггеры

В статье StegoNet [5] предлагается три различных триггера: Логит триггер, Ранговый триггер, Настроенный ранговый триггер.

Метод Логит триггера предполагает запоминание выходов логитов для заранее выбранных входных данных – триггеров. После того, как в модель подается один экземпляр из входных данных, выбранных в качестве триггера, происходит совпадение выходных значений логитов, и вредоносный код извлекается и активируется. Теоретически это невозможно, поскольку вероятность подачи точно такого же входного образца очень мала, а выходы логитов (числа с плавающей точкой) должны полностью совпадать.

Поэтому метод рангового триггера будет более полезен. Разница заключается в сравнении не самих логитов, а их рангов, т.е. индексов в отсортированном массиве логитов. Например, пусть последний слой имеет размерность 3 и триггером будут логиты $\{p_1, p_2, p_3\} = \{0{,}5, 0{,}2, 0{,}4\}$. Но из-за разнообразия входов получили выход $\{0{,}55, 0{,}13, 0{,}42\}$. Ранги логитов будут равны $r = \{p_1, p_3, p_2\}$, и они будут совпадать.

Метод настроенного рангового триггера включает в себя выбор исходной выборки, дополненной различными вариациями, и дообучение модели на этих дополненных выборках. Однако вместо оригинальной функции потерь, зависящей от значений логитов, используется функция потерь, основанная на рангах логитов. Для этого придется вручную задать целевое значение рангов логита. Пусть x – аугментированные входные данные, h^r – установленная для них метка ранга логита, если логит не рассматривается, то его значение равно 0. Функция потерь будет выглядеть следующим образом:

$$arg\min_{w} \frac{1}{n} \sum_{1}^{n} \mathcal{L}(f_w(x), h^r).$$

После активации триггера вредоносная программа собирается в единый код. Затем его хэш-сумма сравнивается с предварительно сохраненной хэш-суммой несегментированной вредоносной программы. Если они совпадают, вредоносная программа запускается.

6.2. Активация

Если к нейросетевой модели прилагается стороннее программное обеспечение, например программа эксплуатации модели, то в нее можно внедрить весь необходимый код для проверки триггеров и активации вредоносного ПО на целевом устройстве. Это самый простой случай, рассмотрим другие.

Сначала предполагалось, что модель обучается на недоверенном источнике, т.е. на стороне злоумышленника, и у него есть все данные модели – атака "белого ящика". Модель передается по сети в сериализованной форме [20], а затем десериализуется. С помощью атаки типа "insecure deserialization" [21, 22] злоумышленник может изменить функции активации в модели. Например, вместо softmax использовать модифицированную версию с проверкой триггеров и развертыванием вредоносных программ.

Другой подход заключается в использовании уязвимостей в библиотеках и исполняемых средах. Например: CVE-2018-6269, CVE-2017-12852.

7. Фреймворк

7.1. Обзор

Целью предложенного автором фреймворка, см. [10], является разработка готового решения для встраивания вредоносных программ в нейронные сети, позволяющего беспрепятственно интегрировать вредоносный код в архитектуру сети. Используя уникальные характеристики нейронных сетей и их широкое применение в различных приложениях, предложенный фреймворк призван исследовать потенциальные риски и уязвимости, связанные с этими моделями.

Предложенный фреймворк представляет собой комплексное решение, позволяющее исследователям и специалистам по безопасности изучать поведение нейронных сетей при воздействии на них встроенного вредоносного ПО. Это открывает возможности для анализа влияния вредоносных атак на производительность сети, выявления уязвимостей и разработки надежных механизмов защиты.

Ключевые особенности:

1. **Внедрение вредоносных программ**. Фреймворк включает в себя различные методы внедрения вредоносных программ в структуру нейронной сети путем изменения их весовых коэффициентов, обеспечивая беспрепятственную интеграцию без ущерба для общей функциональности сети.

2. **Возможность оценки**. Платформа авторов предоставляет инструменты для оценки влияния встроенного вредоносного ПО на производительность сети, т.е. метрики снижения точности.

3. **Гибкость и совместимость**. Фреймворк разработан как совместимый с фреймворком глубокого обучения PyTorch [23], что позволяет легко интегрировать его в существующие архитектуры нейронных сетей без модификации.

В целом платформа авторов позволяет исследователям и практикам в области кибербезопасности лучше понять последствия внедрения вредоносных программ в нейронные сети. Так как платформа надежная и гибкая, то облегчается поиск эффективных мер противодействия и разработка более устойчивых и безопасных моделей нейронных сетей.

7.2. Архитектура и компоненты

Чтобы проверить падение точности моделей до и после внедрения вредоносного ПО с помощью предложенного фреймворка, была проведена серия экспериментов с различными архитектурами нейронных сетей [24].

1. **Набор данных**. Для тестирования модели и структуры был использован набор данных ImageNet [25]. Этот набор содержит 1,2 миллиона трехканальных изображений размером 224×224 пикселей, представляющих 1000 различных категорий объектов.

2. **Методология**.

— Были проведены эксперименты с использованием нескольких предварительно обученных нейросетевых моделей.

— Для каждого эксперимента с помощью предложенного фреймворка были внедрены вредоносные программы разного размера. Примеры вредоносных программ были взяты из репозиториев [16, 26].

— Производительность модифицированных моделей сравнивалась с оригинальными моделями, а также оценивалось влияние внедренных вредоносных программ на точность модели.

3. **Результаты и анализ**.

В табл. 1 показана точность моделей в результате внедрения вредоносных программ с помощью предложенного фреймворка.

Для сравнения возможностей фреймворка были выбраны наиболее часто используемые модели и образцы вредоносного ПО. Поскольку большие модели имеют большую емкость, результаты реализации были показаны только для метода LSB замены, а для остальных методов сравниваются в основном средние и малые модели. Обратите внимание, что для метода устойчивого обучения дообучение моделей не использовалось.

— Результаты эксперимента продемонстрировали эффективность фреймворка при внедрении вредоносных программ в нейронные сети.

— Модифицированные сети успешно сохраняют точность в большинстве случаев классификации на обычных входных данных, демонстрируя желаемое уклончивое поведение.

— Результаты экспериментов выявили компромисс между емкостью моделей и точностью классификации.

Таблица 1. Точность моделей после внедрения вредоносного ПО в их веса с помощью фреймворка. Случаи со значительным снижением точности моделей выделены жирным шрифтом

Метод	Модель	Базовая точность	EquationDrug 372KB	ZeusVM 405KB	NSIS 1,7MB	Mamba 2,30MB	WannaCry 3,4MB	VikingHorde 7,1MB	Artemis 12,8MB
LSB замена	Alexnet	52,8%	52,8%	52,8%	52,8%	52,8%	52,8%	52,8%	52,8%
	Resnet101	76,7%	76,7%	76,7%	76,4%	76,3%	76,2%	75,7%	74,9%
	Inception	68,0%	67,9%	68,0%	68,1%	68,0%	67,9%	67,8%	67,1%
	Resnet50	76,0%	76,0%	76,1%	76,0%	76,3%	75,9%	76,2%	75,2%
	Googlenet	67,1%	66,8%	66,9%	66,7%	65,9%	65,7%	–	–
	Resnet18	67,8%	67,9%	67,8%	67,3%	68,0%	67,5%	**58,4%**	–
	Mobilenet	70,9%	**0,1%**	**0,1%**	**0,1%**	**0,1%**	–	–	–
	Squeezenet	54,9%	**0,1%**	**0,1%**	–	–	–	–	–
MSB сохранение	Inception	68,0%	68,0%	68,2%	67,7%	67,0%	68,1%	61,1%	62,7%
	Resnet18	67,8%	67,7%	67,4%	67,3%	67,0%	66,3%	66,2%	60,9%
	Mobilenet	70,9%	71,1%	69,2%	68,1%	67,0%	63,8%	**0,7%**	–
Быстрая замена	Inception	68,0%	68,0%	67,7%	68,0%	68,1%	67,2%	67,9%	68,0%
	Resnet18	67,8%	67,7%	67,3%	67,2%	67,0%	67,6%	66,2%	61,2%
	Mobilenet	70,9%	70,8%	70,9%	65,7%	**59,8%**	**40,7%**	**1,6%**	–
Половинная замена	Inception	68,0%	68,0%	68,0%	68,0%	68,0%	68,0%	68,0%	68,0%
	Resnet18	67,8%	67,8%	67,8%	67,8%	67,8%	67,8%	67,8%	67,8%
	Mobilenet	70,9%	70,9%	69,9%	69,3%	66,0%	67,7%	**52,0%**	–
Устойчивое обучение	Inception	68,0%	68,4%	67,6%	67,8%	67,3%	68,3%	67,7%	67,8%
	Resnet18	67,8%	67,5%	67,7%	68,0%	67,9%	67,2%	67,3%	68,0%
	Mobilenet	70,9%	**54,9%**	**20,4%**	**0,4%**	**0,4%**	**0,7%**	–	–
Сопоставление значений	Inception	68,0%	68,0%	68,0%	67,4%	67,7%	67,5%	67,2%	66,2%
	Resnet18	67,8%	67,4%	67,4%	67,2%	67,4%	67,9%	67,4%	67,2%
	Mobilenet	70,9%	70,1%	70,7%	**60,1%**	**53,1%**	–	–	–
Сопоставление знаков	Inception	68,0%	68,0%	68,0%	68,0%	–	–	–	–
	Resnet18	67,8%	67,8%	67,8%	67,8%	–	–	–	–
	Mobilenet	70,9%	–	–	–	–	–	–	–

8. Контрмеры

Защита нейронных сетей от внедрения вредоносного ПО имеет решающее значение для обеспечения их целостности и надежности. Для снижения рисков, связанных с этим типом атак [27], можно применить несколько контрмер. Обычно используются следующие контрмеры.

1. **Изменение архитектуры сети**. Можно изменить структуру модели или ее параметры так, чтобы хэш-значение развернутой модели не совпадало с сохраненным. Это может быть достигнуто путем изменения архитектуры сети, добавления случайных слоев или изменения весовых параметров [28].

2. **Использование надежных каналов поставки моделей**. Выбирая надежных и заслуживающих доверия поставщиков MLaaS, организации могут значительно снизить вероятность приобретения зараженных вредоносным ПО моделей [29].

3. **Регулярное обновление моделей**. Для защиты от возникающих угроз очень важно поддерживать модели нейронных сетей в актуальном состоянии с помощью последних патчей и обновлений безопасности. Регулярные обновления моделей с улучшенной архитектурой, надежными алгоритмами обучения и усиленными мерами безопасности помогут предотвратить и обнаружить попытки проникновения вредоносного ПО.

4. **Мониторинг моделей**. Постоянный мониторинг развернутых моделей необходим для выявления необычного поведения или отклонений от ожидаемых моделей. Такие методы, как самоанализ модели, обнаружение аномалий и анализ времени выполнения, помогут выявить потенциальное внедрение вредоносного ПО и инициировать соответствующие ответные меры [30].

9. Заключение

В данной работе представлен новый фреймворк для встраивания вредоносного кода в нейронные сети, использующий весовые параметры нейронов в качестве носителей вредоносной информации. Интегрируя различные методы и компоненты, фреймворк демонстрирует возможность встраивания вредоносного кода в нейросетевые модели, подчеркивая потенциальные риски безопасности, связанные с развертыванием таких моделей.

Экспериментальная оценка этого фреймворка на различных архитектурах нейронных сетей показала его эффективность в успешном внедрении вредоносного ПО при сохранении функциональности и производительности модели. Полученные результаты подчеркивают необходимость разработки надежных мер безопасности для борьбы с растущим уровнем угроз, связанных с внедрением вредоносного ПО в системы машинного обучения.

Проведено обсуждение архитектуры и компонентов этого фреймворка, включая используемые методы внедрения вредоносных программ, процесс интеграции и тестирование модели.

В заключение можно сказать, что данная платформа доказывает наличие уязвимостей в системах машинного обучения и подчеркивает необходимость принятия мер безопасности для обеспечения целостности, надежности и устойчивости этих систем. Понимая и снижая риски, связанные со встроенным вредоносным ПО, можно повысить безопасность приложений машинного обучения и помочь создать более безопасный цифровой ландшафт.

СПИСОК ЛИТЕРАТУРЫ

1. *Namiot D., Ilyushin E., Pilipenko O.* On trusted AI Platforms // Int. J. Open Inform. Techn. 2022. V. 10. No. 7. P. 119–127.

2. *Kostyumov V.* A survey and systematization of evasion attacks in computer vision // Int. J. Open Inform. Techn. 2022. V. 10. No. 10. P. 11–20.
3. *Stoecklin Ph.M., Kirat D., Jang J.* DeepLocker: How AI Can Power a Stealthy New Breed of Malware // SecurityIntelligence. 2018.
4. *Ilyushin E., Namiot D., Chizhov I.* Attacks on machine learning systems-common problems and methods // Int. J. Open Inform. Techn. 2022. V. 10. No. 3. P. 17–22.
5. *Liu T.* StegoNet: Turn Deep Neural Network into a Stegomalware // Annual Computer Security Applications Conference. ACSAC'20. 2020. P. 928–938.
6. *Wang Z.* EvilModel 2.0: Bringing Neural Network Models into Malware Attacks // arXiv:2109.04344. 2021.
7. *Liu T., Wen W., Jin Y.* SIN2: Stealth infection on neural network – A low-cost agile neural Trojan attack methodology // IEEE Int. Symposium on Hardware Oriented Security and Trust. 2018. P. 227–230.
8. *Stefnisson S.* Evasive Malware Now a Commodity // SecurityWeek. 2018.
9. *Bidzhiev T., Namiot D.* Research of existing approaches to embedding malicious software in artificial neural networks // Int. J. Open Inform. Techn. 2022. V. 10. No. 9. P. 21–31.
10. *Bidzhiev T.* NNMalwareEmbedder. 2023. https://github.com/Temish09/NNMalwareEmbedder
11. *Keita K., Michel P., Neubig G.* Weight poisoning attacks on pretrained models // arXiv preprint arXiv:2004.06660. 2020.
12. *Lakshmanan R.* A Large-Scale Supply Chain Attack Distributed Over 800 Malicious NPM Packages // The Hacker News. 2022.
13. *IEEE Computer Society.* IEEE 754-2019 – IEEE Standard for Floating-Point Arithmetic. 2019.
14. *Snehal K., Neeta D. Jacobs D.* Implementation of lsb steganography and its evaluation for various bits // 1st International Conference on Digital Information Management. 2007. P. 173–178.
15. *Howard G.A.* MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications // arXiv:1704.04861. 2017.
16. *ytisf.* theZoo – A Live Malware Repository. 2021. https://github.com/ytisf/theZoo.
17. *Iandola N.F.* SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size // arXiv preprint arXiv:1602.07360. 2016.
18. *Krizhevsky A., Sutskever I., Hinton E.G.* Imagenet classification with deep convolutional neural networks // Advances in neural information processing systems. 2012. No. 25. P. 1097–1105.
19. *Simonyan K., Zisserman A.* Very deep convolutional networks for largescale image recognition // arXiv preprint arXiv:1409.1556. 2014.
20. *Rossum G. van.* pickle – Python object serialization // Python Software Foundation, Python Documentation. 2021.
21. *Trail of Bits.* Fickling. 2021. https://github.com/trailofbits/fickling.
22. *Acunetix.* What is Insecure Deserialization? // Acunetix. 2017.
23. *Paszke A.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019.

24. *Szegedy C.* Going deeper with convolutions // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. P. 1–9.
25. *Deng J.* Imagenet: A large-scale hierarchical image database // IEEE conference on computer vision and pattern recognition. 2009. P. 248–255.
26. *InQuest.* malware-samples. 2021. https://github.com/InQuest/malware-samples.
27. *Yansong G.* Strip: A defence against trojan attacks on deep neural networks // Proceedings of the 35th Annual Computer Security Applications Conference. 2019.
28. *Yansong G.* Backdoor attacks and countermeasures on deep learning: A comprehensive review // arXiv preprint arXiv:2007.10760. 2020.
29. *Parker S., Wu Z., Christofides D.P.* Cybersecurity in process control, operations, and supply chain // Computers & Chemical Engineering. 2023. V. 171. P. 108–169.
30. *Costales R.* Live trojan attacks on deep neural networks // arXiv:2004.11370. 2020

Статья представлена к публикации членом редколлегии А.А. Галяевым.

Поступила в редакцию 08.07.2023

После доработки 24.10.2023

Принята к публикации 20.01.2024