Исследование существующих подходов к встраиванию вредоносного программного обеспечения в искусственные нейронные сети

Т. М. Биджиев, Д. Е. Намиот

***Аннотация*—В последние годы, нейронные сети показывают свой потенциал как новую парадигму решения задач в сфере информационных технологий. Они показали свою эффективность во многих сферах, но обучение нейронных сетей затратно в плане вычислительных ресурсов. В связи с этим, появились услуги обучения сетей на базе облачных технологий, а также получения заранее обученных моделей. Это привнесло новые угрозы для кибербезопасности. Путем разбиения и размещения вредоносного программного обеспечения(ПО) по весовым параметрам нейронов сети, оно может быть передано незаметно, через каналы ненадежных поставщиков услуг. Рассматриваются семь методов внедрения вредоносного ПО и ее активации, включая LSB substitution (замену наименее значащих битов) , Restlience training (устойчивое обучение), Value-mapping (отображение в значения), Sign-mapping (отображение в знаки), MSB reservation (сохранение наиболее значащих битов), Fast substitution (быстрая замена), Half substitution (замена половины). Приводится сравнительный анализ этих методов. Рассматриваются четыре вида триггеров для активации ПО, а именно Знаковый триггер, Logits trigger, Rank trigger, Fine-tuned Rank Trigger. Приводится условный код с реализацией метода LSB замены на языке программирования Python.**

***Ключевые слова*—вредоносное по, нейронные сети, триггеры**

I. Введение

Методы внедрения вредоносного программного обеспечения применяются при поставке готовых моделей конечному потребителю посредством сервисов MLaaS (Machine Learning as a Service). Использование MLaaS означает, что вычисления выполняются за пределами вашей компании – вы платите за необходимые вам услуги машинного обучения, а хранение данных может осуществляться на ваших персональных серверах или в облаке (обычно предлагается поставщиками MLaaS). Примерами могут служить такие сервисы как IBM Watson, Microsoft Azure, Google Cloud ML.

Часто потребители, не являющиеся специалистами в области машинного обучения, работают с серивсами MLaaS без понимания процесса обучения, тестирования, обработки данных. Для таких пользователей наиболее важным критерием является точность модели.

Целью злоумышленника является внедрение вредоносного ПО в модель глубоких нейронных сетей, сохраняя точность модели на приемлемом высоком уровне; передача ПО к конечному потребителю через каналы связи, избежав обнаружения антивирусными программами; запуск вредоносного ПО при выполнении заранее установленного условия активации (триггера).

A. Методология.

На Рис.1 показан общий сценарий взаимодействия пользователя и злоумышленника. Пользователь, который хочет использовать модель нейронных сетей, например, для бизнеса, определяется с архитектурой модели и далее обучает ее посредством сервисов MLaaS, либо загружает уже обученную модель с каких-либо источников. Например, пользователи (и злоумышленники) могут загружать свои модели в платформу ONNX - Open Neural Networx Exchange [3].

В случае, когда злоумышленник выступает в качестве сервиса MlaaS, будем предполагать, что он не может изменять архитектуру полученной сети, но может изменять ее весовые параметры.

В соответствии с Рис.1 для осуществления всех целей злоумышленника, ему необходимо выполнить следующие шаги:

1) Получить от пользователя модель нейронных сетей, либо спроектировать собственную модель;
2) Обучить модель до необходимого уровня точности модели. Это можно сделать самому, либо злоумышленник может сам воспользоваться сервисами MLaaS;
3) Подготовить и Внедрить вредоносное ПО, контролируя потерю точности модели, например, используя методы, рассматриваемые в данной статье. После внедрения, злоумышленнику необходимо оценить модель, чтобы убедиться, что потери точности модели приемлемы. Если потери превышают допустимый диапазон, злоумышленнику необходимо “заморозить” встроенные вредоносные нейроны и дообучить модель, чтобы получить более высокую точность.
4) Создать триггер для развертывания и активации вредоносного ПО, например, используя методы, рассматриваемые в данной статье;
5) Как только модель будет готова, злоумышленник может использовать такие методы, как «загрязнение цепочки поставок»[6], чтобы опубликовать ее в общедоступных репозиториях или других местах.

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

Статья получена 09 августа 2022.

Темирлан Муратбиевич Биджиев, МГУ им. М.В. Ломоносова, (email: temirlanbid@gmail.com).

Дмитрий Евгеньевич Намиот, МГУ им. М.В. Ломоносова, (email: dnamiot@gmail.com).

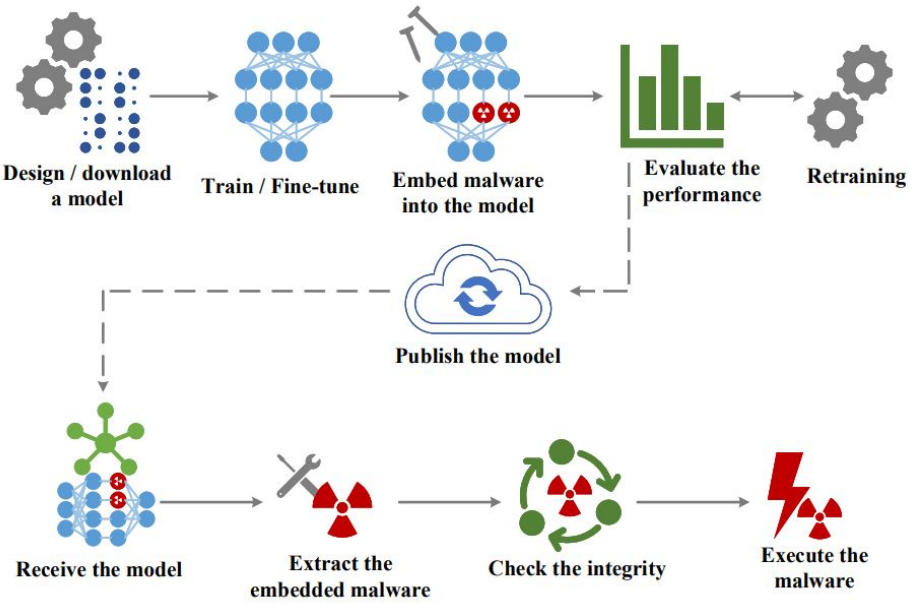


Рис. 1: Общий сценарий взаимодействия пользователя и злоумышленника.

II. Краткое описание модели нейронных сетей.

Модель нейронной сети обычно состоит из входного слоя, нескольких скрытых слоев и выходного слоя. Входной слой принимает входные данные и посылает сигналы на скрытый слой нейронной сети. Нейрон скрытого слоя принимает входящий сигнал от нейронов предыдущего слоя, домноженных на определенные веса, и выводит его на следующий слой после добавления определенного смещения (bias). Выходной слой - это последний слой. Он принимает входящие сигналы от последнего скрытого слоя и нормирует их, чтобы получить выходные данные нейронной сети. Рис. 2

Модели глубоких нейронных сетей демонстрируют высокую эффективность во многих интеллектуальных приложениях. Они состоят из сложных многослойных сетевых структур для обработки входных данных высокой размерности, которые могут быть выражены в виде отображения из пространства входных данных, в пространство ответов (меток) следующим образом:

$$f_w(\cdot) : X \to Y$$

с входными данными $X \in \mathbb{R}^n$, выходными данными $Y \in \mathbb{R}^m$ и параметрами модели (весами) w. Чтобы установить вид этого отображения, модель нейронных сетей обычно строится на различных типах слоев, включая сверточный слой (для извлечения признаков), пулинга (для уменьшения размерности входных данных), функции активации и конечной функции нормализации. Для обучения модели нейронных сетей, параметры w инициализируются случайными значениями, которые будут итеративно обновляться путем минимизации функции потерь L:

$$\arg\min_w \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(f_w(X_n), Y_n),$$

где X_n - это конкретный экземпляр входных данных из обучающего множества, Y_n - соответствующая ей правильна метка, N - размер обучающей выборки, $\mathcal{L}$ - функция ошибки, обычно квадрат разницы результата модели и правильной метки. Это не единственный вид функции потерь, он может варьироваться от решаемой задачи. Задача минимизации может быть решена, например, с помощью алгоритма градиентного спуска. После обучения модель нейронных сетей может быть использована для новых входных данных.

III. Методы внедрения вредоносного ПО.

Нейрон в скрытом слое имеет весовой коэффициент w_i для каждого входного сигнала x_i с предыдущего слоя. Предположим, что все входы нейрона $x = (x_1, x_2, \ldots, x_n)$, а все веса коэффициенты $w = (w_1, w_2, \ldots, w_n)$, где n - количество входных сигналов (т.е. количество нейронов в предыдущем слое). Нейрон получает входной сигнал x и вычисляет скалярное произведение x с весами w. Затем добавляется смещение b. Теперь выходной сигнал нейрона равен $y = f(w \cdot x, b) = f(\sum_{i=1}^{n} w_i x_i, b)$. Мы видим, что каждый нейрон содержит $n+1$ параметров - n весовых параметров (количество нейронов в предыдущем слое) и одно смещение b. Каждый параметр обычно представляет собой 32-битное число с плавающей запятой. Следовательно, размер параметров в каждом нейроне составляет $32(n+1)$ бита, что составляет $4(n+1)$ байта.

A. Внедрение вредоносных байтов в нейроны

По стандарту IEEE[13], число с плавающей точкой имеет размер 32 бита (Рис. 3), где первый бит выделен для знака числа, последующие 8 бит выделены для экспоненты, и последние 23 бита - мантисса. Результатом является число $\pm 1.m \times 2^n$ в двоичном виде, где m - мантисса числа, n - экспонента. Такое число принадлежит диапазону от 2^{-127} до $2^{127} - 1$. За ее порядок отвечает ее экспонента, т.е. сохраняя несколько первых байтов числа, можно оставшуюся часть заменять на байты вредоносного ПО, сохраняя малую разницу с первоначальным числом.

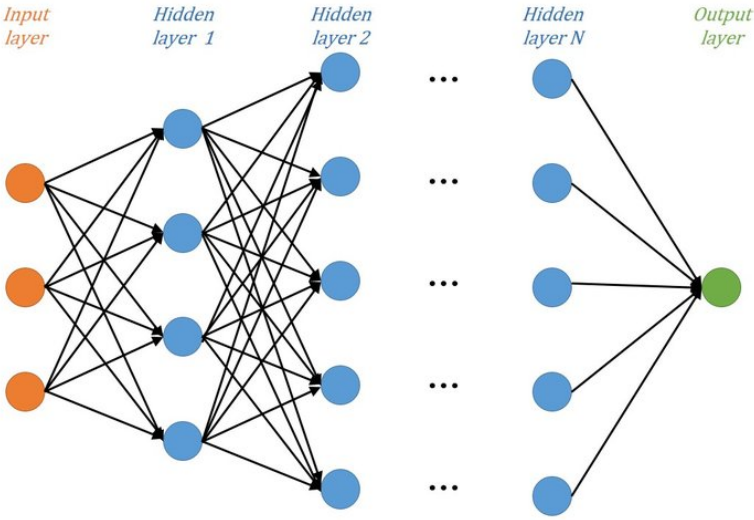


Рис. 2: Структура нейронной сети, состоящая из входного слоя, нескольких скрытых слоев и выходного слоя.

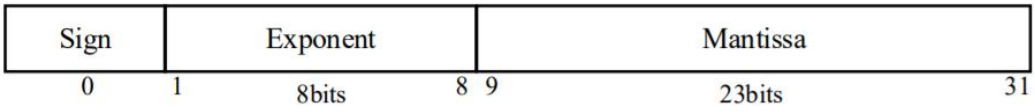


Рис. 3: Формат 32-битного числа с плавающей точкой

B. Методы StegoNet

В статье [8] предложено четыре метода внедрения вредоносного ПО:

- LSB substitution (замена последних по значимости битов);
- Resilience training (Устойчивое обучение);
- Value-mapping (Отображение в значения);
- Sign-mapping (Отображение в знаки);

a) LSB substitution:

Данный метод предполагает LSB замену, применяемую в методах стеганографии. Алгоритм подробно описан в статье [7]. Выбирается количество нейронов и количество параметров, которые будут содержать вредоносное ПО. Вся битовая длина ПО разбивается на части длины количества нейронов и параметров, учитывая выбранную длину LSB подстановки. Эти параметры записываются в несколько первых параметров модели, затем весь код вредоносного ПО записывается в остальные параметры модели. Этот процесс можно представить в виде такого Питон кода. Далее индексация по модели F такова: первый индекс - слой, второй индекс - параметр, третий индекс - битовое строчное представление параметра.

```
import sys

# w: LSB width
# e: end index
# P: binary payloads
# F: binary neural network model

def embedding(w, P, F):
# calculate the number of
# involved parameters
e = sys.getsizeof(P) / w
# store configurations
# we will assume that
# function bin(x) returns binary
# representation of x without prefix '0b'
F[1][1][0:7] = bin(w)
F[1][2][0:15] = bin(e)[16:31]
F[1][3][0:15] = bin(e)[0:15]

# bit substitution with payloads
L = 1; M = 4
W = w - 1 # layers
for i in range(e):
#bit-wise
F[L][M][0:M] = P[w*(i-1) : w*i-1]

# Reached end of layer
if len(F[L]) == M-1:
M = 1

# Move to next layer
L = L + 1
else:
M = M + 1
```

Необходимо оценить вместимость ПО для несжатых и сжатые моделей нейронных сетей, путем измерения размера модели и ее избыточности (т.е. максимальной емкости для внедрения информации без потери точности). Как показано в таблице I, все несжатые модели могут обеспечивать значительную избыточность ($\geq$ 10 МБ), без ущерба для точности, при условии, что вредоносные программы имеют размер в среднем 100 КБ [12]. Простая

Таблица I: Избыточность несжатых и сжатых моделей [8]

Uncompressed DNN Models				
	AlexNet	**GoogLeNet**	**VGG-16**	**ResNet-50**
# Layers	8	22	16	50
# Parameters	61M	7M	138M	25M
Model Size	227MB	27MB	515MB	96MB
# Redundant Bits	21	20	19	16
Redundant Space	152MB	16MB	312MB	47MB

Hardware-favorable Compressed DNN Models				
	Comp.AlexNet	**Comp.VGG-16**	**Mobilenet**	**Squeezenet**
# Layers	8	16	28	18
# Parameters	6.97M	11.26M	4.2M	1.24M
Model Size	6.63MB	10.78MB	4.2MB	4.6MB

“LSB замена”может быть использована для внедрения вредоносного ПО путем замены наименее значащих битов (LSB) параметров нейронной сети двоичным кодом вредоносной программы. Однако такое решение неприменимо к моделям нейронных сетей с высокой степенью сжатия. Как показано в таблице I, размер сжатых моделей был значительно сокращен за счет уменьшения как объема, так и точности данных параметров модели. Например, размер MobileNet [4] составляет всего 4 МБ с 8-битными параметрами в количестве 4 миллионов. Эти сжатые модели быстро теряют точность даже при незначительном изменении параметров. Этот метод не подходит для сжатых моделей.

b) Resilience training:

Удаление некоторого множества нейронов из топологии модели нейронной сети может привести к значительному снижению точности, но параметры, соединяющие оставшиеся нейроны, могут быть подкорректированы (дообучены) до достижения первоначальной точности. Основываясь на этой интуиции, была предложена методика Resilience training (Устойчивое обучение).

Как показано на рисунке 4a, Resilience training состоит из следующих шагов:

1) Вычислить необходимое количество параметров нейронной сети, т.е. $n = \lceil P/q \rceil$, на основе размера вредоносного ПО P и количества параметров q;
2) Произвольно сгенерировать “вектор перестановок индексов чтобы выбрать n параметров случайным образом;
3) Назначить значения сегмента двоичного представления вредоносного ПО каждому выбранному параметру, следуя последовательности в “векторе перестановок индексов” (как вектор $\{A2, B1, \dots\}$ на рисунке 4a);
4) Дообучить модель, фиксируя значения этих параметров.

Метод предполагает прямую замену целых битов выбранных параметров сегментами вредоносного ПО. Такие нейроны (т.е. с измененными параметрами) не будут обновляться во время переобучения. Ожидается, что после обучения точность модели будет восстановлена, что позволит скрыть факт присутствия введенного ПО. В частности, “вектор перестановок индексов” будет сгенерирован именно случайным образом с целью указания выбранных параметров для ввода сегментов кода. Чтобы восстановить разбитый на сегменты код, нужно последовательно объединять двоичный коды каждого параметра, следуя ”вектору перестановок индексов”. Теперь появляется необходимость дополнительно хранить этот вектор.

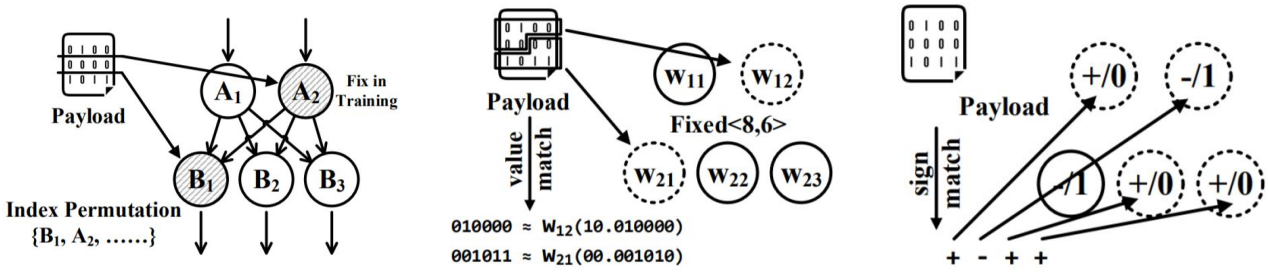
Методы Value mapping и Sign mapping позволяют вводить вредоносные программы в сжатые модели с помощью специальных правил “поиска и сопоставления”и избавляют от процесса дообучения как в методе Resilience training. На рис. 4b и 4c показаны идеи методов.

c) Value mapping:

Т.к. для сжатых моделей часто характерны сжатые параметры, например вместо 32-битных чисел с плавающей точкой, берутся 8-битные числа с фиксированной точкой, то будем рассматривать такие модели. Пусть дана модель с 8-битными числами с фиксированной точкой с 6 битами после запятой. При сопоставлении значений мы сначала разделяем двоичный код вредоносного ПО на сегменты длины количества знаков после запятой. Например, в нашем случае код будет разделен на множество 6-битных сегментов. Затем для каждого сегмента проводится полный исчерпывающий поиск по параметрам модели, чтобы сопоставить (или заменить) одно и то же (или близкое) значение дробных битов параметров. Как пример, показанный на рисунке 4b, для данного сегмента полезной нагрузки “010000” (или “001011”) параметр w_{12} (или w_{21}) соответствует, поскольку значение дробных битов “010000” (или “001010”) совпадает с (или близок с) сегментом полезной нагрузки. Наконец, мы сопоставляем сегмент кода с соответствующим параметром, заменяя дробные биты параметра сегментом кода при необходимости. Часто сжатые модели масштабируются до значений от -1 до 1 поэтому используются только дробные биты при сопоставлении значений. Необходимо сохранять дополнительные вектор параметров, которые мы заменили или сопоставили с битами ПО.

d) Sign mapping:

Метод сопоставления знаков использует аналогичное правило “полного поиска и сопоставления”, основанное на знаковом бите параметров модели. Как показано на рисунке 4c, сопоставление знаков будет проходить через параметры модели и сопоставлять бит знака параметра с каждым отдельным битом в вредоносном коде, например 0 сопоставляется со знаком $+$ параметра, таким образом, в конечном итоге сопоставляя код с последовательностью битов знака для соответствующих параметров. Необходимо сохранять вектор сопоставляемых параметров.



(a) Resilience training

(b) Value mapping

(c) Sign mapping

Рис. 4: Методы внедрения вредоносного ПО [8]

C. Методы статьи EvilModel

В статье [16] проанализированы методы статьи выше, а также предложены три метода, как модификация метода LSB замены:

- MSB reservation (сохранение наиболее значимого байта);
- Fast substitution (быстрая замена);
- Half substitution (Замена половины);

a) MSB reservation:

Поскольку наиболее важная экспоненциальная часть параметра в основном находится в первом байте, первый байт является наиболее значимым байтом для определения значения параметра. Поэтому можно оставить его неизменным и внедрить вредоносное ПО в последующие три байта. Таким образом, значения параметров все еще находятся в разумных пределах. Например, для параметра -0.011762472800910473 (0xbc40b763 в шестнадцатеричном формате), если для последних трех байтов числа заданы произвольные значения (т.е. от 0xbc000000 до 0xfcffffffff), значения параметра будут находиться в диапазоне от -0.0078125 до -0.0312499981374. Мы можем изменить последние три байта параметра на вредоносные байты. Данный подход называется “MSB reservation”.

b) Fast substitution:

Далее, в статье проанализировали распределение параметров одного случайного нейрона модели ImageNet[1]. Если мы установим первый байт параметра равным 0x3C или 0xBC (с единственной разницей в знаковом бите), а остальные биты параметра установим произвольными значениями (т.е. 0x3c000000 до 0x3fffffffff или 0xbc000000 до 0xfcffffffff), значения параметра будут находиться в диапазоне от 0.0078125 до 0.0312499981374, или -0.0312499981374 и -0.0078125. В этой модели 62.65% параметров нейрона попадают в этот диапазон. Поэтому, если заменить параметры тремя байтами вредоносного ПО и первым байтом префикса 0x3C или 0xBC на основе знака параметра, большинство значений параметров все еще будут в разумных пределах. По сравнению с методом MSB reservation, этот метод может оказать большее влияние на производительность модели, но, поскольку ему не нужно разбирать параметры в нейроне, он будет работать быстрее. Метод назвали Fast substitution.

c) Half substitution:

Аналогично MSB reservation, если сохранить первые два байта неизменными, вместо одного, и изменить остальные два байта, значение этого числа будет колебаться в меньшем диапазоне. Например, для параметра 0xbc40b763, если для последних двух байтов установлены произвольные значения (т.е. от 0xbc400000 до 0xbc40ffff), значения находятся в диапазоне от -0.01171875 до -0.0117797842249, разница будет в пределах 10^{-5}. Однако, поскольку в параметре заменяется только два байта, этот метод может внедрять меньше вредоносных программ, чем предыдущие два метода. Метод назвали Half substitution.

IV. Сравнение методов внедрения.

Сравним точности разных моделей до и после внедрения всех методов выше.

Рассмотрим методы статьи StegoNet.

В тиблице II представлена информация о точности тестирования различных моделей нейронных сетей до и после внедрения различных вредоносных программ с использованием метода LSB substitution. Черточка означает, что метод не позволяет внедрить вредоносное ПО в модель из-за ограничения размера. Жирные шрифт означает значительное снижение точности по сравнению с исходной. Данные оценки подвержены ошибкам $< \pm 1\%$ из-за случайности при тестировании нейронных сетей, что объясняет повышение точности в некоторых случаях. Как видно из этой таблицы II, наивный метод LSB замены может поддерживать хорошую точность тестирования на средних нейронных сетях, этот факт не распространяется на малые нейронные сети. Например, это приводит к значительному снижению точности (т.е. к резкому падению до $\approx 0.1\%$) в моделях нейронных сетей с высокой степенью сжатия из-за ограниченной точности данных и уменьшенного количества параметров.

Напротив, Resilience training может относительно лучше поддерживать ввод вредоносного ПО на небольших нейронных сетях, таблица III. Для небольших вредоносных программ, таких как EquationDrug, ZeusVM и Cerber, он может поддерживать точность тестирования на том же уровне, что и оригинал, даже в самой маленькой модели MobileNet (4.2 МБ) и SqueezeNet (4.6 МБ). Однако точность в MobileNet значительно снизилась с 66.7% до 0.7%, поскольку размер вредоносного ПО увеличился с 0.59 МБ (Cerber) до 3.35 МБ (WannaCry). Можно наблюдать, что верхняя граница отношения размера вредоносного кода к размеру модели для метода Resilience training без снижения точности составляет $\approx 15\%$.

Однако такая проблема была устранена с помощью методов, основанного на “полном поиске и отображе-

Таблица II: Точности моделей до и после внедрения вредоносного ПО методом LSB substitution. Жирным выделены случаи с существенным падением точности модели. [8]

LSB Substitution

		Original	EquationDrug	ZeusVM-decrypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	74.6%	75.7%	75.5%	75.8%	74.9%	74.5%	76.1%	75.6%	75.3%	74.7%
	Inceptionv3	78%	78.2%	77.9%	76.8%	76.3%	78%	77.2%	78.3%	78.2%	78.1%	77.3%
	Densnet201	77%	77.3%	77.1%	76.5%	77.3%	75.9%	76.3%	77.6%	77%	76.4%	77.1%
	Resnet18	70.7%	69.3%	71.2%	71.1%	70.2%	70.5%	71.6%	72.1%	71.3%	69.3%	**61.3%**
Small DNNs	GoogLeNet	69.8%	68.4%	68.1%	70.3%	70.8%	69.7%	69.4%	68.7%	69%	**58.1%**	**55.3%**
	Comp.VGG-16	70.5%	**51.6%**	**32.1%**	**17.3%**	**7.5%**	**0.9%**	**0.1%**	**0.2%**	**0.1%**	**0.1%**	-
	Comp.AlexNet	57%	**31.2%**	**17.6%**	**0.2%**	**0.1%**	**0.1%**	**0.1%**	**0.1%**	**0.1%**	-	-
	SqueezeNet	57.5%	**0.7%**	**0.3%**	**0.2%**	**0.1%**	**0.1%**	**0.1%**	**0.2%**	**0.1%**	-	-
	MobileNet	70.9%	**0.2%**	**0.2%**	**0.1%**	**0.1%**	**0.1%**	**0.2%**	**0.1%**	**0.2%**	-	-

Таблица III: Точности моделей до и после внедрения вредоносного ПО методом Resilience training. Жирным выделены случаи с существенным падением точности модели. [8]

Resilience Training

		Original	EquationDrug	ZeusVM-decrypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	75.2%	75.4%	74.8%	74.7%	75.1%	75.3%	74.6%	74.8%	75.5%	75.1%
	Inceptionv3	78%	78.3%	78.4%	78.2%	77.9%	78.4%	78.1%	77.6%	78.4%	77.8%	78.1%
	Densnet201	77%	77.2%	76.7%	77.1%	76.9%	77.3%	76.5%	77.2%	77.3%	76.7%	76.4%
	Resnet18	70.7%	71.1%	71.2%	70.9%	71%	70.4%	70.3%	70.9%	71.3%	68.2%	69.7%
Small DNNs	GoogLeNet	69.8%	70.3%	69.2%	69.6%	71%	70.5%	69.3%	70.4%	70.2%	70.4%	68.2%
	Comp.VGG-16	70.5%	68.3%	69.1%	71.2%	69.1%	68.4%	**63.4%**	**56.1%**	**22.6%**	**6.7%**	-
	Comp.AlexNet	57%	55.4%	56.7%	57.2%	54.1%	**38.2%**	**34.3%**	**16.7%**	**3.9%**	-	-
	SqueezeNet	57.5%	56.8%	54.3%	53.2%	**48.3%**	**35.4%**	**29.6%**	**15.1%**	**4.1%**	-	-
	MobileNet	70.9%	71.2%	68.5%	66.7%	**54.4%**	**32.5%**	**29.1%**	**6.1%**	**0.7%**	-	-

Таблица IV: Точности моделей до и после внедрения вредоносного ПО методом Value mapping. Жирным выделены случаи с существенным падением точности модели. [8]

Value-Mapping

		Original	EquationDrug	ZeusVM-decrypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	74.8%	74.7%	75.1%	75.3%	74.6%	74.8%	74.6%	75.7%	75.5%	75.8%
	Inceptionv3	78%	78.3%	78.4%	78.2%	78%	77.2%	78.3%	78.4%	78.1%	77.6%	77.3%
	Densnet201	77%	77.1%	76.9%	77.3%	76.5%	77.2%	76.5%	77.3%	75.9%	77.3%	76.7%
	Resnet18	70.7%	71.1%	70.2%	70.5%	71.6%	72.1%	70.9%	71%	70.4%	70.3%	70.9%
Small DNNs	GoogLeNet	69.8%	70.1%	68.3%	70.2%	68.1%	70.3%	69.8%	68.4%	68.1%	70.3%	70.8%
	Comp.VGG-16	70.5%	71.3%	71%	68.3%	69.1%	71.2%	69.1%	69.2%	**48.7%**	-	-
	Comp.AlexNet	57%	56.9%	56.7%	57.2%	54.1%	-	-	-	-	-	-
	SqueezeNet	57.5%	57.3%	56.9%	55.7%	56.8%	**39.7%**	**43.2%**	**21.8%**	-	-	-
	MobileNet	70.9%	69.2%	71%	70.5%	70.3%	**54.7%**	**48.6%**	**49.3%**	-	-	-

нии", таблица IV. Для сильно сжатых моделей нейронных сетей, таких как "Comp.Alexnet", параметры модели чрезвычайно сжаты, метод может быть менее эффективным для таких моделей, т.е. параметры также сжаты. Аналогичную тенденцию можно обнаружить и в методе Sign mapping, таблица V. Однако в целом сопоставление знаков всегда может поддерживать первоначальную точность тестирования для всех применимых случаев.

Оценим методы статьи EvilModel.

Результат метода MSB reservation показан в таблице VI. Из-за избыточности моделей нейронных сетей, когда вредоносная программа внедрена, точность тестирования не влияет на модели большого размера ($>$ 200 МБ). Точность немного возросла в некоторых случаях (например, Vgg 16 с NSIS, Inception с Nimda и Googlenet с Eternal Rock), как также отмечено в статье Stegonet [8]. При внедрении с использованием MSB reservation точность снижается с увеличением размера встроенного вредоносного ПО для моделей среднего и малого размера. Например, точность снижается на 5% для моделей среднего размера, таких как Resnet101 с Lazarus, Inception с Lazarus и Resnet50 с Mamba. Теоретически, максимальный порядок встраивания (т.е. отношение размера вредоносного кода к размеру модели) метода MSB reservation составляет 75%. В эксперименте верхнюю границу порядка встраивания без значительного снижения точности составляет 25.73% (Googlenet с Artemis).

В таблице VII приведен результат метода Fast substitution. Производительность метода аналогична методу MSB reservation, но нестабильна для небольших моделей. Когда более крупная вредоносная программа внедряется в модель среднего или малого размера, производительность значительно снижается. Например, для Googlenet с Lazarus точность тестирования резко падает до 0.526%. Существуют также случаи повышения точности, такие как Vgg19 с NSIS. Это показывает, что быстрая замена может быть использована в качестве замены методу MSB reservation, когда модель большая или задача требовательна ко времени. В эксперименте порядок встраивания получился, без значительного снижения точности, 15.9% (Resnet18 с Viking Horde).

Метод Half substitution превосходит все другие методы, как показано в таблице VIII. Из-за избыточности моделей нейронных сетей точность после внедрения кода всех размеров практически не ухудшается, даже когда почти половина модели была заменена вредоносными байтами, точность колеблется в пределах 0.01% от оригинала. Небольшой размер Squeezenet (4.74 МБ) может встроить образец вируса Mamba объемом 2.3 МБ с точностью, увеличивающейся на 0.048%. Теоретически, максимальный порядок встраивания составляет 50%. В эксперименте достигалось близкое к теоретическому значение в 48.52% (Squeezenet с Mamba).

V. Методы активации вредоносного ПО.

A. История развития вредоносного ПО.

Давайте кратко рассмотрим эволюцию уклончивых вредоносных программ [10]:

- В конце 1980-х и начале 1990-х годов первые варианты полиморфных и метаморфных вирусов были

Таблица V: Точности моделей до и после внедрения вредоносного ПО методом Sign mapping. Жирным выделены случаи с существенным падением точности модели. [8]

		Sign-Mapping										
		Original	EquationDrug	ZeusVM-decrypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Large DNNs	Vgg19(17.16MB)	71.1%	71.3%	71.2%	70.7%	71.2%	70.9%	71.1%	71.2%	71.1%	70.8%	71.3%
	Vgg16(16.45MB)	70.5%	71%	70.6%	69.9%	70.2%	71.1%	70.8%	70.2%	69.8%	69.7%	71%
	AlexNet(7.27MB)	57%	57.2%	57.1%	56.7%	57.3%	57.2%	56.8%	56.9%	57.3%	-	-
	ResNet101(5.32MB)	77.1%	77.2%	77.4%	76.7%	76.3%	77%	77.3%	77.5%	-	-	-
Medium DNNs	Resnet50(3.05MB)	75.2%	74.8%	75.3%	75.1%	75.5%	75.4%	75.3%	74.9%	-	-	-
	Inceptionv3(2.84MB)	78%	77.4%	78.2%	78.1%	77.8%	78%	78.1%	-	-	-	-
	Densnet201(2.38MB)	77%	77.3%	77.2%	76.4%	76.7%	77.1%	76.4%	-	-	-	-
	Resnet18(1.39MB)	70.7%	71.1%	70.8%	71%	71.2%	69.5%	-	-	-	-	-
Small DNNs	GoogLeNet(0.83MB)	69.8%	68.3%	70.2%	70.1%	-	-	-	-	-	-	-
	Comp.VGG-16(1.34MB)	70.5%	71.3%	71%	70.4%	69.6%	-	-	-	-	-	-
	Comp.AlexNet(0.83MB)	57%	55.4%	56.7%	56.3%	-	-	-	-	-	-	-
	SqueezeNet(0.15MB)	57.5%	-	-	-	-	-	-	-	-	-	-
	MobileNet(0.5MB)	70.9%	68.3%	-	-	-	-	-	-	-	-	-

Таблица VI: Точности моделей до и после внедрения вредоносного ПО методом MSB reservation. Жирным выделены случаи с существенным падением точности модели. [16]

MSB Reservation	Vgg19 548.14MB	Vgg16 527.87MB	AlexNet 233.1MB	Resnet101 170.45MB	Inception 103.81MB	Resnet50 97.75MB	Googlenet 49.73MB	Resnet18 44.66MB	Mobilenet 13.55MB	Squeezenet 4.74Mb
Base	74.218%	73.360%	56.518%	77.374%	69.864%	76.130%	62.462%	69.758%	71.878%	58.178%
EternalRock, 8KB	74.216%	73.360%	56.516%	77.366%	69.868%	76.120%	62.472%	69.604%	71.814%	58.074%
Stuxnet, 24.4KB	74.222%	73.354%	56.520%	77.370%	69.862%	76.148%	62.400%	69.742%	71.778%	57.630%
Nimda, 56KB	74.224%	73.362%	56.516%	77.350%	70.022%	76.112%	62.402%	69.746%	71.584%	56.640%
Desover, 89.7KB	74.220%	73.360%	56.512%	77.384%	69.900%	76.040%	62.354%	69.702%	71.232%	58.838%
OnionDuke, 123.5KB	74.224%	73.368%	56.502%	77.362%	69.780%	76.092%	62.282%	69.678%	71.440%	52.314%
Mirai, 175.2KB	74.218%	73.366%	56.516%	77.352%	69.832%	76.146%	62.256%	69.670%	71.186%	54.442%
Turla, 202KB	74.214%	73.352%	56.502%	77.336%	69.860%	76.048%	62.392%	69.756%	70.964%	52.774%
Jigsaw, 283.5KB	74.228%	73.372%	56.486%	77.328%	69.666%	75.990%	62.150%	69.664%	71.032%	50.814%
EquationDrug, 372KB	74.198%	73.370%	56.504%	77.296%	69.912%	76.026%	62.062%	69.672%	70.880%	**42.488%**
ZeusVM, 405KB	74.210%	73.360%	56.490%	77.280%	69.756%	76.028%	61.956%	69.568%	71.108%	**43.774%**
Electro, 598KB	74.218%	73.348%	56.484%	77.288%	69.832%	75.990%	62.082%	69.562%	67.138%	**36.018%**
Petya, 788KB	74.240%	73.382%	56.478%	77.242%	69.402%	75.898%	61.430%	69.486%	66.910%	**11.772%**
NSIS, 1.7MB	74.250%	73.390%	56.446%	77.164%	68.128%	75.800%	61.486%	69.238%	68.496%	**25.624%**
Mamba, 2.30MB	74.212%	73.350%	56.466%	77.082%	67.754%	75.672%	62.102%	69.108%	**61.488%**	**2.606%**
WannaCry, 3.4MB	74.210%	73.372%	56.446%	76.976%	65.966%	75.642%	60.898%	68.926%	65.292%	**0.304%**
Pay2Key, 5.35MB	74.206%	73.358%	56.498%	76.936%	68.668%	75.440%	62.138%	68.549%	**12.048%**	-
VikingHorde, 7.1MB	74.214%	73.350%	56.436%	76.734%	68.094%	75.064%	61.132%	67.000%	**0.218%**	-
Artemis, 12.8MB	74.190%	73.364%	56.408%	74.386%	61.086%	70.408%	61.196%	**58.734%**	-	-
Lazarus, 19.94MB	74.180%	73.342%	56.376%	71.412%	**56.388%**	**63.698%**	**58.124%**	**37.582%**	-	-

Таблица VII: Точности моделей до и после внедрения вредоносного ПО методом Fast substitution. Жирным выделены случаи с существенным падением точности модели. [16]

MSB Reservation	Vgg19 548.14MB	Vgg16 527.87MB	AlexNet 233.1MB	Resnet101 170.45MB	Inception 103.81MB	Resnet50 97.75MB	Googlenet 49.73MB	Resnet18 44.66MB	Mobilenet 13.55MB	Squeezenet 4.74Mb
Base	74.218%	73.360%	56.518%	77.374%	69.864%	76.130%	62.462%	69.758%	71.878%	58.178%
EternalRock, 8KB	74.216%	73.360%	56.516%	77.366%	69.870%	76.120%	62.462%	69.754%	71.818%	58.074%
Stuxnet, 24.4KB	74.222%	73.354%	56.520%	77.370%	69.868%	76.148%	62.462%	69.742%	71.748%	57.630%
Nimda, 56KB	74.224%	73.362%	56.516%	77.350%	69.870%	76.112%	62.462%	69.746%	71.570%	56.640%
Destover, 89.7KB	74.220%	73.360%	56.512%	77.384%	69.874%	76.040%	62.462%	69.702%	71.314%	56.838%
OnionDuke, 123.5KB	74.224%	73.368%	56.502%	77.362%	69.890%	76.092%	62.462%	69.678%	71.460%	52.314%
Mirai, 175.2KB	74.218%	73.366%	56.516%	77.352%	69.930%	76.146%	62.462%	69.670%	71.090%	54.442%
Tuna, 202KB	74.214%	73.352%	56.502%	77.336%	69.862%	76.048%	62.462%	69.756%	70.932%	52.774%
Jigsaw, 283.5KB	74.228%	73.372%	56.486%	77.328%	69.966%	75.990%	62.462%	69.664%	70.976%	51.364%
EquationDrug, 372KB	74.198%	73.370%	56.504%	77.296%	69.916%	76.026%	62.462%	69.672%	71.038%	**42.648%**
ZeusVM, 405KB	74.210%	73.360%	56.490%	77.280%	69.898%	76.028%	62.462%	69.568%	71.142%	**41.144%**
Electro, 598KB	74.218%	73.348%	56.484%	77.288%	69.880%	75.990%	62.462%	69.562%	67.106%	**14.822%**
Petya, 788KB	74.240%	73.382%	56.478%	77.242%	69.924%	75.898%	62.462%	69.486%	67.094%	**6.912%**
NSIS, 1.7MB	74.250%	73.390%	56.466%	77.164%	69.528%	75.800%	62.462%	69.238%	68.496%	**10.318%**
Mamba, 2.30MB	74.212%	73.350%	56.466%	77.082%	69.556%	75.672%	62.462%	69.108%	**60.564%**	**0.814%**
WannaCry, 3.4MB	74.210%	73.372%	56.446%	76.976%	69.092%	75.642%	62.462%	68.926%	**24.262%**	**0.100%**
Pay2Key, 5.35MB	74.206%	73.358%	56.498%	76.936%	68.594%	75.440%	62.462%	68.340%	**0.192%**	-
VikingHorde, 7.1MB	74.214%	73.350%	56.436%	76.734%	64.682%	75.074%	62.462%	67.350%	**0.108%**	-
Artemis, 12.8MB	74.190%	73.364%	56.408%	74.502%	61.252%	70.062%	**51.256%**	60.272%	-	-
Lazarus, 19.94MB	74.180%	73.342%	56.376%	70.720%	**54.470%**	**59.490%**	**0.526%**	**20.882%**	-	-

разработаны для разрушения и уничтожения данных. С помощью запутывания и изменения полезных нагрузок авторы вредоносных программ избегали антивирусных систем, которые могли легко проверять файлы на наличие известных шаблонов с использованием статических сигнатур. Следовательно, антивирусная индустрия постепенно развивала возможности статического кода и анализа вредоносных программ для анализа запутанного кода и определения злонамеренных намерений кода или файлов, запущенных на защищаемых ими конечных точках.

- В 1990-х годах авторы вредоносных программ начали шифровать вредоносную полезную нагрузку (используя так называемые упаковщики), чтобы вредоносный код был виден только тогда, когда он был расшифрован в памяти перед его выполнением. Индустрия безопасности отреагировала динамическим анализом вредоносных программ, создав начальные версии ”песочниц”для вредоносных программ, таких как виртуализированные системы, в которых запускаются подозрительные исполняемые

Таблица VIII: Точности моделей до и после внедрения вредоносного ПО методом Half substitution. Жирным выделены случаи с существенным падением точности модели. [16]

MSB Reservation	Vgg19 548.14MB	Vgg16 527.87MB	AlexNet 233.1MB	Resnet101 170.45MB	Inception 103.81MB	Resnet50 97.75MB	Googlenet 49.73MB	Resnet18 44.66MB	Mobilenet 13.55MB	Squeezenet 4.74Mb
Base	74.218%	73.360%	56.518%	77.374%	69.864%	76.130%	62.462%	69.758%	71.878%	58.178%
EternalRock, 8KB	74.218%	73.358%	56.522%	77.374%	69.864%	76.130%	62.464%	69.760%	71.878%	58.178%
Stuxnet, 24.4KB	74.216%	73.360%	56.520%	77.374%	69.862%	76.130%	62.462%	69.762%	71.880%	58.176%
Nimda, 56KB	74.220%	73.360%	56.522%	77.372%	69.862%	76.132%	62.460%	69.754%	71.880%	58.176%
Destover, 89.7KB	74.218%	73.362%	56.522%	77.376%	69.862%	76.130%	62.458%	69.756%	71.882%	58.162%
OnionDuke, 123.5KB	74.218%	73.360%	56.522%	77.376%	69.864%	76.128%	62.456%	69.762%	71.874%	58.168%
Mirai, 175.2KB	74.220%	73.362%	56.526%	77.372%	69.864%	76.130%	62.454%	69.758%	71.874%	58.174%
Tuna, 202KB	74.218%	73.360%	56.526%	77.372%	69.864%	76.132%	62.464%	69.764%	71.882%	58.168%
Jigsaw, 283.5KB	74.216%	73.358%	56.522%	77.382%	69.862%	76.126%	62.458%	69.750%	71.886%	58.170%
EquationDrug, 372KB	74.220%	73.362%	56.522%	77.378%	69.862%	76.132%	62.464%	69.744%	71.884%	58.176%
ZeusVM, 405KB	74.218%	73.362%	56.520%	77.378%	69.862%	76.132%	62.460%	69.762%	71.884%	58.176%
Electro, 598KB	74.220%	73.354%	56.526%	77.376%	69.860%	76.124%	62.456%	69.742%	71.878%	58.168%
Petya, 788KB	74.214%	73.358%	56.522%	77.376%	69.864%	76.128%	62.444%	69.738%	71.886%	58.190%
NSIS, 1.7MB	74.220%	73.368%	56.522%	77.372%	69.854%	76.126%	62.442%	69.756%	71.890%	58.192%
Mamba, 2.30MB	74.222%	73.342%	56.520%	77.368%	69.868%	76.118%	62.452%	69.756%	71.888%	58.226%
WannaCry, 3.4MB	74.226%	73.364%	56.512%	77.370%	69.864%	76.128%	62.462%	69.758%	71.880%	-
Pay2Key, 5.35MB	74.222%	73.348%	56.510%	77.368%	69.864%	76.124%	62.452%	69.778%	71.890%	-
VildngHorde, 7.1MB	74.222%	73.358%	56.490%	77.354%	69.866%	76.122%	62.450%	69.740%	-	-
Artemis, 12.8MB	74.220%	73.358%	56.506%	77.374%	69.874%	76.122%	62.448%	69.718%	-	-
Lazarus, 19.94MB	74.228%	73.362%	56.512%	77.350%	69.870%	76.136%	62.422%	69.710%	-	-

файлы (называемые образцами), отслеживаются их действия.

- В 2000-х годах первые формы уклончивого вредоносного ПО — вредоносного ПО, пытающегося активно избегать анализа. Например, вредоносная программа использовала проверки, чтобы определить, работает ли она в виртуализированной среде и присутствуют ли другие процессы, которые, как известно, выполняются в изолированных средах вредоносных программ, чтобы избежать анализа и сохранить свои секреты в зашифрованном виде. Этот подход все еще распространен сегодня, поскольку исследование, проведенное в мае 2018 года на SecurityWeek [14], показало, что 98 процентов проанализированных образцов вредоносных программ в той или иной степени используют методы уклонения.

Тем не менее, хотя защита от вредоносных программ продолжает развиваться, даже самые последние формы целевых вредоносных программ требуют предопределенных триггеров, которые могут быть обнаружены защитниками путем проверки кода, файлов конфигурации или сетевой активности. Все эти триггеры видны опытным аналитикам вредоносных программ с соответствующими инструментами.

Рассмотрим эти триггеры.

B. Триггеры

В статье EvilModel разрабатывается триггер на основе знаков значений предпоследнего слоя. Пусть, у модели на предпоследнем слое находится 128 нейронов, которые производят 128 выходных сигналов. Будем преобразовывать этот выходной вектор сигналов t в вектор $v = (v_i = sign(t_i))$, т.е. вектор из нулей и единиц - значения знаков элементов вектора t. Далее этот вектор проходит через хэширования и получается 32-значная строка. Далее эта строка сравнивается с заранее сохраненной хэш-строкой триггером. При полном совпадении вредоносное ПО активируется.

В качестве триггера к сборке и запуску вредоносного ПО могли бы послужить какие-то шаблоны на входных данных (измененный пиксель, и т.п.), но обрабатывать входные сигналы трудоемко. В дополнении к этому, из-за внешнего шума входных данных, данный триггер не всегда будет правильно срабатывать. Вместо этого, можно брать выводы логитов последнего слоя, т.е. выводов до нормализации(например функцией softmax). Их преимуществом является то, что их размер обычно во много раз меньше размера входного слоя (для ImageNet в 154 раза).

в статье StegoNet предложены три разных триггера:

- Logits trigger;
- Rank trigger;
- Fine-tuned Rank Trigger.

Суть метода Logits Trigger состоит в том, чтобы запомнить выводы логитов для определенных входных данных - триггеров. Тогда, когда на вход подадут тот же триггер, выводы логитов совпадут и вредоносный код соберется и будет активирован.

В теории такое невозможно, т.к. крайне мал шанс подачи абсолютно того же входного сэмпла, а выводы логитов (числа с плавающей точкой) должны совпадать полностью.

Поэтому более полезным будет метод Rank trigger. Различие заключается в том, чтобы вместо выводов логитов сравнивать их ранки, т.е. индексы в отсортированном массиве логитов. Например, пусть последний слой имеет размерность 3 и вывод логитов равен $\{p_1, p_2, p_3\} = \{0.5, 0.2, 0.3\}$. Но из-за разнообразия входных данных, мы получили вывод $\{0.55, 0.13, 0.32\}$. Ранки логитов будут равны $r = \{p_1, p_3, p_2\}$, и они совпадут.

Данный метод более надежный, однако бывают случаи сильно аугментированных тех же самых триггерных входных данных, что ранки перестают совпадать. Здесь поможет третий метод Fine-tuned Rank Trigger.

Сначала выбирается триггерный входной сэмпл, затем он аугментируется на разные вариации, и на них модель будет дообучаться, но с тем отличием, что вместо оригинальной функции потерь, зависящей от значений логитов, мы возьмем функцию потерь от ранков логитов. Для этого придется вручную установить значение метки. Пусть x - аугментированные входные данные, h^r - установленная к ним метка ранков логитов, по такому правилу: сумма чисел вектора равна 1, если логит не рассматривается его значение устанавливается в 0. Например,

для ожидаемого вектора ранков $\{1, -, 3, 2\}$ с 4 логитами, и вторым ненужным логитом, $h^r = \{0.5, 0, 0.17, 0.33\}$. Функция потерь будет выглядеть так:

$$\arg\min_w \frac{1}{n} \sum_1^n \mathcal{L}(f_w(x), h^r).$$

После того как триггер сработал, вредоносное ПО собирается в единый код. Затем его хэш-сумма сравнивается с заранее сохраненной хэш-суммой несегментированного вредоносного кода. Если они совпадают, вредоносный код запускается.

C. Запуск

Если модель нейронной сети сопровождается сторонним ПО, например, программой эксплуатации модели, то в нее можно внедрить весь необходимы код проверки триггеров и активации вредоносного ПО на целевом устройстве.

В начале мы предположили, что модель тренируется в непроверенном источнике, т.е. у злоумышленника, и у него все данные модели, атака white box. Модель передается по сети в сериализованном виде, затем десериализуется. Благодаря атаке как insecure deserialization [9], злоумышленник может изменять функции активации в модели. Например, вместо softmax использовать модифицированную версию с проверкой на триггеры и развертыванием вредоносного ПО.

Другим подходом может служить использование уязвимостей библиотек и исполняющих сред. Например, в таблице IX указаны некоторые из уязвимостей, которые могут быть использованы для активации ПО. Например, рассмотрим несколько CVE.

CVE-2018-6269. NVIDIA Jetson TX2 содержит уязвимость в драйвере ядра, из-за которой обработка управления вводом/выводом (IOCTL) для запросов пользовательского режима может привести к разыменованию ненадежного указателя, что может привести к раскрытию информации, отказу в обслуживании, повышению привилегий или выполнению кода.

CVE-2017-12852. В функции numpy.pad в Numpy 1.13.1 и более ранних версиях отсутствует проверка ввода. Пустой список или ndarray застрянут в бесконечном цикле, что может позволить злоумышленникам вызвать DoS-атаку.

VI. Контрмеры

Когда нейроны заменяются вредоносными байтами, структура модели остается неизменной. Поскольку вредоносная программа разделяется по нейронам, ее характеристики больше недоступны, это позволяет избежать обнаружения антивирусными системами.

В качестве контрмер к подобным атакам можно отнести следующие методы:

- Изменение архитектуры сети. Для активации вредоносного ПО, необходимо, чтобы хэш-значение развернутого ПО совпало с сохраненным, поэтому путем ручного изменения архитектуры сети, его весовых параметров, или сжатия модели, можно избежать его активации.
- Использование сжатых моделей. Как было показано в статье, чем больше модель сжата, тем меньше видов вредоносного ПО можно в него внедрить, поэтому использование таких моделей уменьшает шанс получения вредоносной модели.
- Использование надежных каналов поставки моделей. Можно избежать получение вредоносного ПО заранее выбирая доверенных поставщиков услуг MLaaS.

VII. Заключение

В настоящей работе рассмотрены различные подходы для внедрения вредоносного ПО в искусственные нейронные сети. В них входят такие методы, как LSB substitution, Resilience training, Value-mapping, Sign-mapping, MSB reservation, Fast substitution, Half substitution. Был проведен сравнительный анализ этих методов, включая темпы падения точностей моделей после внедрения ПО. Были рассмотрены различные триггеры для активации ПО, в том числе Logits trigger, Rank trigger, Fine-tuned Rank Trigger. Данная обзорная статья является подспорьем для дальнейшего изучения различных аналогичных подходов, и для создания методов защиты от подобных угроз. В качестве дальнейшей работы можно

- проанализировать зависимость падения точности модели от внедрения вредоносного ПО в различные слои ИНС;
- проанализировать темпы обнаружения измененных моделей существующими антивирусами в зависимости от размера вредоносного ПО;
- увеличить робастность методов внедрения ПО относительно изменения весовых параметров модели;
- провести сравнительный анализ триггеров;
- разработать контрмеры к рассмотренным в статье методам.
- разработать методы обнаружения вредоносного ПО в моделях нейронных сетей.

Благодарность

Мы благодарны сотрудникам кафедры Информационной безопасности факультета ВМК МГУ имени М.В. Ломоносова за ценные обсуждения данной работы.

Исследование выполнено при поддержке Междисциплинарной научно-образовательной школы Московского университета «Мозг, когнитивные системы, искусственный интеллект»

Статья является продолжением серии публикаций, посвященных устойчивым моделям машинного обучения и кибербезопасности систем искусственного интеллекта [11], [5], [2]. Она подготовлена в рамках проекта кафедры Информационной безопасности факультета ВМК МГУ имени М.В. Ломоносова по созданию и развитию магистерской программы ”Искусственный интеллект в кибербезопасности”[15].

Список литературы

[1] Jia Deng и др. «Imagenet: A large-scale hierarchical image database». В: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, с. 248—255.

Таблица IX: Уязвимости. [8]

Common Vulnerabilities and Exposures				Other approaches	
DNN Software	**Component**	**CVE ID**	**Type**	**Component**	**Type**
Jetson TX2	Kernel	CVE-2018-6269	execution	Python	Insecure API
Jetson TX2	Kernel	CVE-2018-6269	execution	Javascript	Obfuscation
Caffe/Torch	Jibjasper	CVE-2017-9782	Overflow	JSON	Insecure Deserialization
Caffe/Torch	OpenCV	CVE-2016-1516	execution	XML	
Tensorflow	Numpy	CVE-2017-12852	DOS	Pickle	
Tensorflow	Wave	CVE-2017-14144	DOS	Protocol Buffer	

[2] Namiot Dmitry и Eugene Ilyushin. *"Generative Models in Machine Learning."* International Journal of Open Information Technologies 10.7 (2022): 101-118.

[3] The Linux Foundation. *ONNX*. url: https://onnx.ai/ (дата обр. 21.05.2022).

[4] Andrew G. Howard и др. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». В: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. url: http://arxiv.org/abs/1704.04861.

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

[6] The Hacker News - Ravie Lakshmanan. *A Large-Scale Supply Chain Attack Distributed Over 800 Malicious NPM Packages*. url: https://thehackernews.com/2022/03/a-threat-actor-dubbed-red-lili-has-been.html (дата обр. 19.04.2022).

[7] Tao Liu, Wujie Wen и Yier Jin. «SIN²: Stealth infection on neural network — A low-cost agile neural Trojan attack methodology». В: *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2018, с. 227—230. doi: 10.1109/HST.2018.8383920.

[8] Tao Liu и др. «StegoNet: Turn Deep Neural Network into a Stegomalware». В: *Annual Computer Security Applications Conference*. ACSAC '20. Austin, USA: Association for Computing Machinery, 2020, с. 928—938. isbn: 9781450388580. doi: 10.1145/3427228.3427268. url: https://doi.org/10.1145/3427228.3427268.

[9] PortSwigger Ltd. *Insecure deserialization*. url: https://portswigger.net/web-security/deserialization (дата обр. 22.05.2022).

[10] Dhilung Kirat Marc Ph. Stoecklin co-authored by Jiyong Jang. *DeepLocker: How AI Can Power a Stealthy New Breed of Malware*. 2018. url: https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/ (дата обр. 12.02.2022).

[11] Eugene Ilyushin Namiot Dmitry и Oleg Pilipenko. *"On trusted AI Platforms."* Internation Journal of Open Information Technologies 10.7 (2022): 119-127.

[12] Yuval Nativ. *theZoo - A Live Malware Repository*. 2015. url: https://thezoo.morirt.com/ (дата обр. 22.05.2022).

[13] IEEE Computer Society. *IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic*. 2019. url: https://standards.ieee.org/ieee/754/6210/ (дата обр. 11.04.2022).

[14] Siggi Stefnisson. *Evasive Malware Now a Commodity*. 2018. url: https://www.securityweek.com/evasive-malware-now-commodity (дата обр. 22.05.2022).

[15] *Искусственный интеллект в кибербезопасности*. https://cs.msu.ru/node/3732, проверено: 08.08.2022.

[16] Zhi Wang и др. *EvilModel 2.0: Bringing Neural Network Models into Malware Attacks*. 2021. arXiv: 2109.04344 [cs.CR].

Research of existing approaches to embedding malicious software in artificial neural networks

Temirlan Bidzhiev, Dmitry Namiot

***Abstract*—In recent years, neural networks have shown their potential as a new paradigm for solving problems in the field of information technology. They have shown their effectiveness in many areas, but training neural networks is expensive in terms of computing resources. In this regard, there are services for training networks based on cloud technologies, as well as obtaining pre-trained models. This has introduced new threats to cybersecurity. By splitting and placing malicious software to the weight parameters of the neurons of the network, it can be transmitted imperceptibly, through the channels of unreliable service providers. Seven methods of malware introduction and activation are considered, including LSB substitution (replacement of the least significant bits), Restlience training, Value-mapping, Sign-mapping, MSB reservation (saving the most significant bits), Fast substitution, Half substitution. A comparative analysis of these methods is given. Four types of triggers for software activation are considered, namely the Sign trigger, Logits trigger, Rank trigger, Fine-tuned Rank Trigger. The code with the implementation of the LSB substitution method in the Python programming language is given.**

***Keywords*—malware, neural networks, triggers**

References

[1] Jia Deng et al. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[2] Namiot Dmitry and Eugene Ilyushin. *"Generative Models in Machine Learning."* International Journal of Open Information Technologies 10.7 (2022): 101-118.

[3] The Linux Foundation. *ONNX*. url: https://onnx.ai/ (visited on 05/21/2022).

[4] Andrew G. Howard et al. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. url: http://arxiv.org/abs/1704.04861.

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

[6] The Hacker News - Ravie Lakshmanan. *A Large-Scale Supply Chain Attack Distributed Over 800 Malicious NPM Packages*. url: https://thehackernews.com/2022/03/a-threat-actor-dubbed-red-lili-has-been.html (visited on 04/19/2022).

[7] Tao Liu, Wujie Wen, and Yier Jin. «SIN²: Stealth infection on neural network — A low-cost agile neural Trojan attack methodology». In: *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2018, pp. 227–230. doi: 10.1109/HST.2018.8383920.

[8] Tao Liu et al. «StegoNet: Turn Deep Neural Network into a Stegomalware». In: *Annual Computer Security Applications Conference*. ACSAC '20. Austin, USA: Association for Computing Machinery, 2020, pp. 928–938. isbn: 9781450388580. doi: 10.1145/3427228.3427268. url: https://doi.org/10.1145/3427228.3427268.

[9] PortSwigger Ltd. *Insecure deserialization*. url: https://portswigger.net/web-security/deserialization (visited on 05/22/2022).

[10] Dhilung Kirat Marc Ph. Stoecklin co-authored by Jiyong Jang. *DeepLocker: How AI Can Power a Stealthy New Breed of Malware*. 2018. url: https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/ (visited on 02/12/2022).

[11] Eugene Ilyushin Namiot Dmitry and Oleg Pilipenko. *"On trusted AI Platforms."* Internation Journal of Open Information Technologies 10.7 (2022): 119-127.

[12] Yuval Nativ. *theZoo - A Live Malware Repository*. 2015. url: https://thezoo.morirt.com/ (visited on 05/22/2022).

[13] IEEE Computer Society. *IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic*. 2019. url: https://standards.ieee.org/ieee/754/6210/ (visited on 04/11/2022).

[14] Siggi Stefnisson. *Evasive Malware Now a Commodity*. 2018. url: https://www.securityweek.com/evasive-malware-now-commodity (visited on 05/22/2022).

[15] *Artificial intelligence in cybersecurity*. https://cs.msu.ru/node/3732, checked: 08.08.2022.

[16] Zhi Wang et al. *EvilModel 2.0: Bringing Neural Network Models into Malware Attacks*. 2021. arXiv: 2109.04344 [cs.CR].