公開されたAIモデルに潜むリスクと新たな攻撃手法

若井 琢朗[1,a)]　戸田 宇亮[1,4,b)]　久保 佑介[2,c)]　森 達哉[1,3,4,d)]

概要：本論文は，公開されることを前提としたAIモデルに対する新たな攻撃手法として，「Weights Injection 攻撃」と「Tensor Trigger 攻撃」を提案し，その有効性を実証することを狙いとする．Weights Injection 攻撃は，深層学習モデルの重みに悪意あるコードを埋め込み，推論時にそのコードが実行されるようにする手法であり，具体的には，精度に影響しない範囲でモデルを構成する約120万個のパラメータのうち，わずか36個を書き換えることで攻撃が成功することを明らかにした．一方，Tensor Trigger 攻撃は，特定のトリガー画像を入力することで悪意あるコードが実行される攻撃手法であり，生成されたトリガー画像によって攻撃が実行される確率が高かったが，特に24ビット幅のコードを用いた攻撃では成功率が向上することが確認された．体系的な実験評価の結果，これらの攻撃手法が高い確率で成功し，たとえばC2エージェント攻撃では640,000ビットのコード埋め込みにもかかわらず，推論精度に影響を与えずに攻撃が成功することが示された．また，これらの攻撃がアンチウィルスソフトウェアの検知を回避するステルス性も有していることを確認した．最後に，これらの攻撃に対する防御手法や検知手法についても議論する．

On the Risks Inherent in Publicly Released AI Models and New Attack Techniques

Takuro Wakai[1,a)]　Takaaki Toda[1,4,b)]　Yusuke Kubo[2,c)]　Tatsuya Mori[1,3,4,d)]

Abstract: This paper aims to propose and demonstrate the effectiveness of two novel attack techniques targeting AI models intended for public release: the “Weights Injection Attack” and the “Tensor Trigger Attack.” The Weights Injection Attack involves embedding malicious code into the weights of a deep neural network model, with the key finding that altering as few as 36 out of approximately 1.2 million parameters can successfully execute the attack without affecting model accuracy. On the other hand, the Tensor Trigger Attack involves executing malicious code through the input of a specific trigger image. It was observed that the attack had a high success rate, particularly when using 24-bit wide code, which further improved the success rate. Systematic experimental evaluation demonstrated that these attack methods are highly effective, as evidenced by the success of the C2 agent attack, which embedded 640,000 bits of code without compromising inference accuracy. Additionally, it was confirmed that these attacks possess stealth capabilities that allow them to evade detection by antivirus software. Finally, the paper discusses potential defense mechanisms and detection methods to counter these attacks.

1. はじめに

深層ニューラルネットワーク（Deep Neural Network, 以下DNN）は，現代社会において多様なシステムに統合され，不可欠な技術として広く認識されている．しかしながら，この普及に伴い，DNNの脆弱性が甚大な被害をもたらす可能性も増大している．そのため，DNNのセキュリティ強化は喫緊の課題となっている．

そのような背景の下，DNNに対する多様な攻撃手法とその防御策が研究されてきた．代表的な攻撃手法としては敵対的攻撃 [1] やデータ汚染攻撃 [2] などが挙げられる．これらの攻撃の主たる脅威は，モデルに誤った推論を引き起

[1] 早稲田大学 / Waseda University
[2] NTT コミュニケーションズ株式会社 / NTT Communications
[3] 情報通信研究機構 / NICT
[4] 理化学研究所 革新知能統合研究センター / RIKEN AIP
[a)] wakataku@nsl.cs.waseda.ac.jp
[b)] todatakaaki@nsl.cs.waseda.ac.jp
[c)] yuusuke.kubo@ntt.com
[d)] mori@seclab.jp

© 2024 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

こすことであり，従来から広く研究が進められてきた．しかしながら，これらの攻撃は本質的に間接的な性質を有している．DNN の利用そのものでは直接的な被害は発生せず，誤った推論結果が他のシステムに利用されることで初めて被害が顕在化するためである．

一方で，近年の高精度な DNN への需要増加に伴い，学習データの収集や学習コストは増大の一途を辿っている．これらのコスト削減を目的として，外部で訓練されたモデルの利用が一般化しつつある．例えば GitHub や Hugging Face などのプラットフォームを通じて，学習済みモデルがインターネット上で公開され，広く利用可能となっている．また，Machine Learning as a Service (MLaaS) などのサービスを通じて外部に学習を委託することも可能となっている．これらの外部モデルの利用は新たなセキュリティリスクを生み出している．第三者によって作成されたモデルの安全性確認が不十分な場合，悪意のあるモデルによる攻撃のリスクが生じる．現状では，悪意ある DNN モデルの危険性が十分に認識されておらず，多くの利用者がその安全性の確認を怠っている状況にある．

本研究では，前述した状況（AI モデルに対する直接的な攻撃に関する評価の不在，学習済みの公開 AI モデルの普及とリスク）の下，公開された DNN モデルの利用自体が攻撃の起点となる新たな攻撃ベクトルとして，「Weights Injection 攻撃」と「Tensor Trigger 攻撃」を提案する．これらの攻撃は，DNN の推論を行うだけで攻撃者の意図した悪性コードが自動的に実行されるという，従来にない直接的な脅威をもたらす．

Weights Injection 攻撃では，DNN モデルの重みに悪性コードを埋め込み，推論時にユーザのシステム上でコードが再構築され実行される．一方，Tensor Trigger 攻撃では，攻撃者が準備したトリガー画像を入力することで，推論計算中に悪性コードが実行される．これらの攻撃手法は，PyTorch のモデル保存形式や Tensor の特性を巧みに利用し，高いステルス性と実行の容易さを両立している．本研究の主な貢献は以下の通りである：

- DNN モデルに対する新たな攻撃ベクトルとして，Weights Injection 攻撃と Tensor Trigger 攻撃を提案し，その実現可能性を実証した．具体的には，Weights Injection 攻撃では約 120 万個のパラメータのうちわずか 36 個の書き換えで攻撃が成功し，Tensor Trigger 攻撃では Tensor の特性を利用して効率的にコードを埋め込むことに成功した．
- 提案された攻撃手法が高い成功率を持つことを示した．例えば，C2 エージェント攻撃では 640,000 ビットのコード埋め込みにもかかわらず，推論精度に影響を与えずに攻撃が成功することを実証した．
- 提案された攻撃が Windows Defender などのアンチウィルスソフトウェアの検知を回避する高いステルス性を有することを示した．
- これらの攻撃に対する防御手法として，ファインチューニングなどの具体的な対策を提示し，DNN セキュリティにおける新たなリスクを提起しつつ，潜在的な被害の最小化に貢献した．

本論文では，提案された攻撃手法の詳細な構築方法と仮想環境での実験結果を提示する．さらに，攻撃の実行の容易さとステルス性の評価，両攻撃手法の比較，および潜在的な対策についての議論を展開する．これらの成果は，DNN のセキュリティに関する新たな知見を提供し，今後の AI システムの安全性向上に寄与するものと期待される．

2. 背景

本章では，本研究で提案する攻撃手法の基盤となる PyTorch によるモデルの取り扱い，及び機械学習モデルが公開されているプラットフォームについて示す．PyTorch は，提案する攻撃手法の実装に用いられており，その特性を理解することが攻撃メカニズムの理解に不可欠である．また，公開プラットフォームの存在は，攻撃の影響範囲を拡大させる要因となる．

2.1 PyTorch

2.1.1 PyTorch の概要

PyTorch は，機械学習と深層学習のためのオープンソースのライブラリであり，主に Python 言語で使用される．強力なテンソル計算機能と自動微分システムを備えており，ニューラルネットワークの設計とトレーニングを容易にする．本研究では，PyTorch の柔軟性と広範な普及を考慮し，攻撃手法の実装基盤として選択した．

2.1.2 PyTorch のモデル定義

PyTorch では，ニューラルネットワークモデルは torch.nn.Module のサブクラスとして定義される．最も重要なメソッドは以下の 2 つである：

- **__init__メソッド：** モデルのレイヤーとパラメータの初期化を行う．
- **forward メソッド：** 入力データを受け取り，ネットワークを通して前方伝播させて出力を生成する．

基本的にはこの 2 つのメソッドのみでモデルが定義されるが，その中身は自由にコードを書くことが可能である．例えば，forward メソッド内に if 文などを追加することにより，柔軟なモデルを作成可能である．この自由度の高さが，推論に関係ないコードの挿入を可能にし，本研究で提案する攻撃手法の基礎となる．

2.1.3 PyTorch のモデル保存

DNN の学習は多くの時間とリソースを必要とするため，一度学習したらその結果を保存することが一般的である．PyTorch では，モデルの保存方法として主に次の 3 つが提供されている．

© 2024 Information Processing Society of Japan

(1) **モデルの state_dict を保存：** state_dict とは，モデルのパラメータ（重みとバイアス）を含む辞書である．最も安定的で公式に推奨されている方法だが，モデルのロード時に毎回モデルクラスを定義し直す必要がある．
(2) **モデル全体を保存：** このアプローチでは, state_dict とモデルの全体構造を保存する．これにより，モデルをロードする際にモデル構造を再定義する必要が無い．ただし，実際にはモデルクラスそのものが直接保存されている訳ではなく，モデルクラスを内包するファイルへの参照パスのみが記録される．そのため，モデルのロード時にはモデルクラスが記載されたファイルが特定の場所に必要であり，環境の変化に非常に弱い．
(3) **TorchScript 形式でモデル保存：** TorchScript は，PyTorch モデルを静的なグラフに変換するための PyTorch モデルの中間表現である．これにより，モデルを Python の実行環境に依存しない形式で保存し，さまざまなプラットフォームでの展開が容易になる．唯一この方法ではロード時にモデルクラスを必要としない．

これらの保存方法の特性は，本研究で提案する攻撃手法の設計と実装に重要な影響を与える．特に，TorchScript 形式での保存は，攻撃コードの埋め込みと実行において中心的な役割を果たす．

3. 攻撃手法

本章では，提案する 2 つの攻撃の手法と，攻撃シナリオを示す．

3.1 Weights Injection 攻撃

3.1.1 概要

本攻撃のアイデアは，ニューラルネットワーク内の重みパラメータのうち，精度に影響しない数ビット程度の領域に攻撃コードを埋め込むことである．それを図解したものが図 1 である．埋め込まれたコードは，推論の際に再構築されて実行される．コードはバラバラにして埋め込まれるため，アンチウィルスによる検知は非常に困難である．しかし, モデルクラス内に再構築と実行のコードを記述する必要があるため，攻撃を隠蔽するためにはモデルクラスの中身をユーザから隠す工夫が必要である．

3.1.2 攻撃手順

以下に本攻撃の手順を示す．
(1) 再構築・実行のコードを含んだモデルを定義する
(2) モデルの学習を行う
(3) 攻撃コードを分散させて重みパラメータに埋め込む
(4) 重みを含んだモデル全体を保存する
(5) モデルを公開，またはユーザに配布する
通常のモデルの学習と異なるのは，手順 1 と 3 のみである．

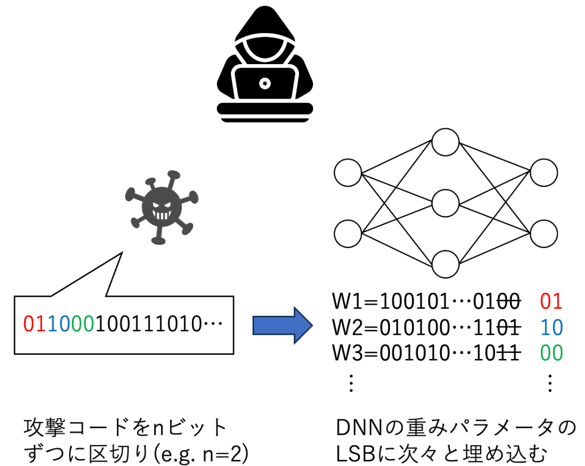

図 1 Weights Injection 攻撃の概要

本攻撃の核となる埋め込みと再構築・実行について以下の項で詳細に説明する．

3.1.3 攻撃コードの埋め込み

本項では，モデルの重みパラメータに攻撃コードを埋め込む方法を説明する．埋め込みは正常に学習されたパラメータに対して攻撃者の手元で行われる．埋め込みについて特筆すべき点は，攻撃コードを分割した点と，それらをパラメータの Least Significant Bit (LSB) に埋め込んだ点である．まず前者は，アンチウィルスの検知を回避する目的である．後者は，モデルの推論精度への影響を最小限に抑え，攻撃のステルス性を上げるためである．モデルのパラメータの書き換えは，その推論精度が低下する可能性が高い．しかし書き換え箇所を LSB にすることで，精度劣化を回避することができる．

具体的には，攻撃コードの長さと攻撃コード自体を結合したものを，区切り幅 w_1 ビットずつに区切る．それを学習済みモデルの重みパラメータの各数値の LSBw_1 ビットに次々と埋め込む．また, 攻撃コードの長さを同時に埋め込むことで，全パラメータのうちどの部分が攻撃コードに該当するか特定する．

3.1.4 攻撃コードの再構築・実行

本項では，モデルの重みパラメータから攻撃コードを再構築・実行する手順の詳細を説明する．ユーザが推論を行った際に攻撃コードが実行される必要があるため，モデルクラス内にその動作を記述する．

再構築フェーズでは，埋め込みと逆のことを行い，攻撃コードを復元する．最初に攻撃コードの長さを取得し，その長さ分だけ重みパラメータからコード断片を取り出す．実行フェーズでは，取り出したバイト列を一旦実行ファイルとして書き出し，実行してから削除するという手順を取る．

3.1.5 モデルの保存方法

2.1.3 項にて，PyTorch には 3 種類のモデル保存方法が

© 2024 Information Processing Society of Japan

あることを述べた．方法 3 は, モデルクラスのファイルをユーザの手元に配置する必要がなく，本攻撃の根幹となる再構築・実行のコードをユーザから隠蔽できるため, 最もステルス性が高いと言える．しかし，再構築フェーズにおいて, モデルの重みパラメータを取得する攻撃コードが原因で TorchScript 形式へのコンパイルが失敗するため, 方法 3 は使用できない．したがって, Weight Injectionk 攻撃では次点でステルス性の高い方法 2 を採用する．

3.2 Tensor Trigger 攻撃

3.2.1 概要

本攻撃のアイディアは，モデルによる推論計算過程において攻撃コードが生成されるような入力データ（トリガー）を予め計算しておくことにある．攻撃者は，ユーザにそのようなトリガーデータを入力させることにより，生成された攻撃コードを実行させることが可能である．それを図解したものが図 2 である．トリガー入力（ここでは画像），およびモデルのパラメタを詳細に調査しても，攻撃の存在を明らかにすることは困難であり，画像をモデルに入力することで初めて成立する攻撃である．

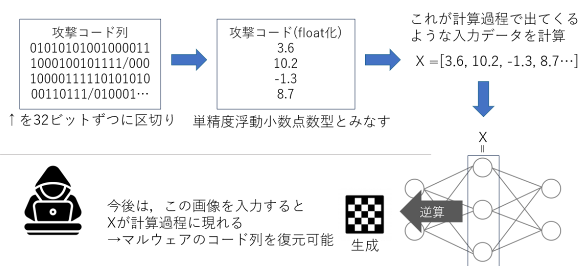

図 2 Tensor Trigger 攻撃の概要

3.2.2 攻撃手順

以下に本攻撃の手順を示す．

(1) 再構築・実行のコードを含んだモデルを定義する
(2) モデルの学習を行う
(3) 攻撃コードのバイナリを特定のビット幅に分割し，各要素が単精度浮動小数点数型の Tensor X に変換する
(4) 中間層の結果が Tensor X と一致するようにモデルを逆算し，入力画像を生成する
(5) モデルを公開，またはユーザに配布し，その生成画像を入力させる

本攻撃ではモデル自体に施す細工は, 再構築・実行のコードを入れるだけで，パラメータなどは正常に学習されたままである．そのため通常の入力においては完全に正常な振る舞いをする．なお実際の攻撃では，ユーザにトリガー画像の入力を促すために，テスト用画像やトラブルシューティング用の画像などと偽ってモデルの仕様書に記載する．本攻撃の核となる画像生成と再構築・実行について以下の項で詳細に説明する．

3.2.3 画像生成方法

本項では，推論の計算過程に攻撃コードが現れるような入力画像を生成する方法を説明する．画像生成は正常に学習されたモデルに対して攻撃者の手元で行われる．ここで，攻撃者は第 n 層と第 $n+1$ 層の間の計算結果 Tensor X_n に攻撃コードが現れるように設定するものとする．

(1) 攻撃コードの長さと攻撃コード自体を結合したものを 32 ビットずつに区切り，それぞれを単精度浮動小数点数型とみなす
(2) それらの数値を並べて Tensor をつくる
(3) Tensor X_n がその値と一致するようにモデルを逆算
(4) 逆算結果をピクセル値とみなして画像に変換

ここでは簡単のため，32 ビットずつに区切っているが，逆算過程の誤差の影響を受けやすいので，例えば 20 ビットずつに区切って残りを 0 埋めして誤差を吸収することも考えられる．

3.2.4 攻撃コードの再構築・実行

以下に攻撃コードを再構築する具体的な手順を示す．

(1) 入力がトリガー画像である場合は以下を実行
(2) 計算過程で出てくる Tensor X_n の各要素を 32 ビットのバイナリ表現とみなす
(3) 先頭から攻撃コードの長さを取り出し，その分だけバイナリ表現を結合

なお，実行手順は 3.2.4 項と同様である．

3.2.5 モデルの保存方法

本攻撃で作成するモデルは TorchScript 形式に変換可能であるため，2.1.3 項にて説明した 3 つの保存方法のうち最もステルス性が高い方法 3 を採用する．

3.3 攻撃シナリオ

本研究では 2 種類の攻撃を提案しているが，図 3，図 4 に示した通りどちらの攻撃も大まかな流れは共通であり，以下の 3 ステップからなる．

(1) 攻撃者が細工したモデルや入力画像を準備する
(2) ユーザに配布する
(3) ユーザの手元で攻撃コードが実行される

ステップ 1，3 は各攻撃の詳細にて説明した通りである．ここで，ステップ 2 のユーザに配布とは，以下のいずれかの方法により配布されることを想定している [3].

- **正規 DNN モデルを装ってインターネット上に公開：** この脅威モデルは，攻撃者は偽の学習済みモデルをインターネット上に公開し，ユーザがこれをダウンロードして使用することによって被害を受けるというものである．前節で挙げたプラットフォーム上に公開されることが多い．
- **学習の委託先が攻撃者の場合：** 学習の委託先が悪意を持った人間である場合は，攻撃用に細工された DNN モデルを作成される可能性があ

© 2024 Information Processing Society of Japan

る．直接の委託先が善良でも，サプライチェーン上に悪意のある人間が存在すれば脅威となる．

図 3，図 4 には前者の方法で書かれている．

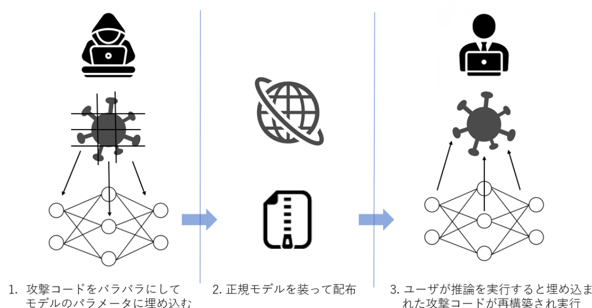

図 3 Weights Injection 攻撃の流れ

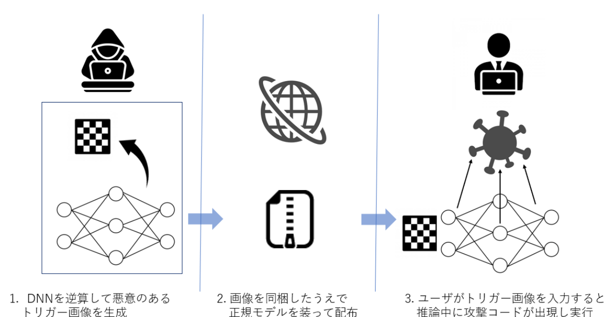

図 4 Tensor Trigger 攻撃の流れ

4. 実験

本章では，攻撃の検証のために実施した実験について示す．

4.1 実験環境

本実験では，攻撃用とユーザ用として 2 つの仮想マシン（VM）を用意した．表 1 にそれぞれのマシンの情報を示す．マルウェア検体の入手の都合上，今回は Windows マシンへの攻撃を実行する．

4.2 DNN モデル

実験に使用した DNN モデルは，MNIST の分類を学習させた CNN モデルである．図 5 にそのモデル構造を示す．なお，テストデータによる分類精度は 99.12%であった．

4.3 マルウェア

2 種類のマルウェア検体を使用する．なお，これらは研究用途に限定したサンプル検体である．1 つ目は fork 爆弾 [4] と呼ばれる非常に軽量な DoS 攻撃の一種である．プロセスの自己複製を高速で繰り返すことでシステムのプロセスリストを埋め尽くし，新たなプロセスを生成不能にする．なお，サイズは 6 バイトである．2 つ目は，Havoc [5] と呼ばれる Command & Control (C2) 用の OSS で使用されている C2 エージェントの実行バイナリである．なお，サイズは 80k バイトほどである．

2 つの攻撃それぞれについて，これらの 2 つのマルウェアで攻撃の成功可否を検証する．マルウェアを 2 種類用いるのは，その長さによる攻撃可能性の変化を観察するためである．

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MyNet                                    [32, 10]                  --
├─Conv2d: 1-1                            [32, 32, 26, 26]          320
├─Conv2d: 1-2                            [32, 64, 24, 24]          18,496
├─MaxPool2d: 1-3                         [32, 64, 12, 12]          --
├─Dropout: 1-4                           [32, 64, 12, 12]          --
├─Linear: 1-5                            [32, 128]                 1,179,776
├─Dropout: 1-6                           [32, 128]                 --
├─Linear: 1-7                            [32, 10]                  1,290
==========================================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 385.63
==========================================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 15.01
Params size (MB): 4.80
Estimated Total Size (MB): 19.91
==========================================================================================
```

図 5 実験で使用した CNN モデル

4.4 実験方法

4.4.1 実験全体の手順

実験方法としては，両攻撃とも以下の手順に従う．

(1) 攻撃者側のマシンで細工された DNN モデルを作成
(2) scp でモデル（と入力画像）をユーザ側に送信
(3) ユーザ側で推論を実行

1 の詳細な手順は 3 章で説明した通りである．ただし，攻撃者が任意に決めるパラメータの値は以下の節で示す．

4.4.2 パラメータ設定

Weights Injection 攻撃における攻撃コード分割幅は 2 ビットとし，攻撃コードの長さは 24 ビットで表す．一方 Tensor Trigger 攻撃における攻撃コードの分割幅は 32 ではなく 24 ビットとし，攻撃コードの長さは同様に 24 ビットで表す．そして攻撃コードが現れるのは 1 層目の計算が終了した直後ということになる．これは n の値が小さいほど逆算が容易で，攻撃の成功率が上がると考えられるからである．図 5 から分かる通り，1 層目は畳み込み層でもともと 28x28 のピクセルサイズであった MNIST の画像を 3x3 のフィルタで畳み込みを行い，結果として 26x26 の Tensor を得ている．この 26x26 の部分に意図したコードが出現するように入力画像を生成することになる．

4.5 評価方法

攻撃に対する評価は以下の 3 点に着目する．

- 攻撃の成功可否
- Windows Defender による検知を回避可能か
- 埋め込みによって推論精度が変化するか

© 2024 Information Processing Society of Japan

表 1　実験で使用したマシン情報

使用者	OS	スペック	メモリ
攻撃者	Ubuntu 20.04.6 LTS	Intel(R) Xeon(R) CPU E3-1230 v3, 3.30GHz	3 GB RAM
ユーザ	Windows10 Home	Intel(R) Xeon(R) CPU E3-1230 v3, 3.30GHz	8 GB RAM

5. 結果

本章では，実験の結果とその評価を示す．

5.1 全体の結果

まずは全体の結果を表 2 に示す．この表における各行は，4.5 節で定義された本実験の評価基準である．各基準が達成された項目は「○」で表され，達成されていない項目は「×」で示されている．

表 2　各攻撃のシミュレーション結果

	Weights Injection 攻撃		Tensor Trigger 攻撃	
	fork 爆弾	C2	fork 爆弾	C2
成功可否	○	○	○	×
検知回避	○	○	○	×
推論精度	○	○	○	○

5.2 Weights Injection 攻撃

5.2.1 fork 爆弾

fork 爆弾のバイナリは 6 バイト (=48 ビット) であるため，2 ビットずつに分割すると 24 個に分かれる．攻撃コードの長さを表す 24 ビットは 12 個に分かれるため，合計で 36 個のパラメータを書き換えることになる．図 5 で示した通り，このモデルは合計で約 120 万個のパラメータを持っているため，書き換えを行うパラメータはごくわずかであることがうかがえる．

攻撃は成功し，推論と同時にユーザの下で fork 爆弾が起動した．攻撃コードを埋め込んだモデルパラメータや，実行時に生成されたバイナリはともに Windows Defender の検知を回避することに成功している．そして，推論精度にも一切影響を与えず，99.12%を保ったままであった．

5.2.2 C2 エージェント

C2 エージェントは 80k バイト (=640,000 ビット) であるため，2 ビットずつに分割し，その長さの情報も付加すると合計で約 32 万個のパラメータを書き換えることになる．fork 爆弾と比較すると非常に多いが，モデル全体のパラメータ数と比較すると 1/4 ほどで済んでいる．

攻撃は成功し，推論と同時にユーザと攻撃者の間に C2 通信が確立した．こちらも同様に Windows Defender の検知を回避することに成功している．そして，大量のパラメータを書き換えたため推論精度の低下を危惧していたが，推論精度には一切影響を与えず，99.12%を保ったままであった．これは恐らくこのモデルが MNIST の分類という比較的簡単なタスクをさせるには十分複雑な構造であったからであると考えられる．そしてこれは，さらに大きな攻撃コードも問題なく埋め込める可能性も示唆している．

5.3 Tensor Trigger 攻撃

5.3.1 fork 爆弾

今回の分割幅は 24 ビットであるため，48 ビットの攻撃コードは 2 つに分割されることになり，その長さの情報 24 ビットも付加すると合計で 3 個の数値が指定されることになる．Tensor X_n の 26x26 個の数値の内 3 個のみを指定するため非常にアンバランスなように思えるが，分割幅を小さくして攻撃コードを分散させると逆算過程での誤差の影響を受けやすくなることが実験から判明し，上記設定が最も成功率が高いことが分かっている．攻撃のために生成された画像は図 6 の左に示す．なお，図中の目盛はピクセルの座標を表している．

攻撃は成功し，推論と同時にユーザの下で fork 爆弾が起動した．攻撃コードを埋め込んだモデルパラメータや，実行時に生成されたバイナリはともに Windows Defender の検知を回避することに成功している．なお，この攻撃はモデルパラメータの書き換えを行わないため，推論精度の低下は一切発生しない．

5.3.2 C2 エージェント

C2 エージェントは 80k バイト (=640,000 ビット) であるため，24 ビットずつに分割し，その長さの情報も付加すると合計で約 2.7 万個の途中計算の値が必要である．しかし，Tensor X_n は 26x26 (=676) 個の数値しか保持していないため，1 枚の画像で指定できる値も最大で 676 個である．そのため複数の画像を生成し，マルウェアのコード列を少しずつ出現させるようにする必要がある．最終的にはそれらをすべて結合して実行させることになる．なお，攻撃のために生成された画像 (1 枚目) は図 6 の右に示す．

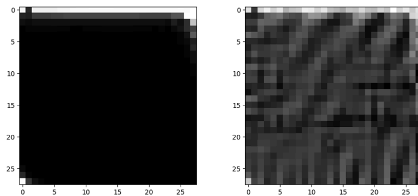

図 6　生成された入力画像（左: fork 爆弾，右: C2）

結果としては，この攻撃は成功しなかった．それは逆算の誤差によるもので，意図した途中計算の再現が困難だからである．その確認のため，攻撃コードから算出した

© 2024 Information Processing Society of Japan

Tensor X_n と，逆算によって生成された画像を入力したときの Tensor X_n の各要素の排他的論理和を取った結果を図 7 に示す．理想的には，これらの 2 つの Tensor は一致しているため，その排他的論理和はすべて 0 になっているはずである．しかし，実際は 1 も多く存在しており，これは意図したコードからビットが反転していることを意味する．本攻撃はビット単位の精度が要求されるため，1 ビットでも反転すると正しい結果が得られない．このようなビット反転が多数発生しているため，攻撃の有効性が大きく損なわれる結果となった．

```
[['00000000000000000000000000000000',
  '00000000000000000000000000000000',
  '00000000000000000000000000000000',
  '00000001111111111111111111111101',
  '00000000000000000000000000000001',
  '00000000000000000000000000000000',
  '00000000000000000000000000000000',
  '00000000000000000000000000000000',
  '00000000000000000000000000000001',
  '00000001111111111111111111111110',
  '00000000000000000000000000000000',
  '00000000000000000000000000000000',
  '00000000000000000000000000000010',
  '00000000000000000000000000000011',
```

図 7 指定したバイナリ列と攻撃時に現れるバイナリ列との排他的論理和（一部抜粋）

6. 議論

本章では本攻撃への対策・検知手法及び今後の課題について論じる．

6.1 対策

本攻撃への対策として，防御手法と検知手法についてそれぞれ考察する．

6.1.1 防御手法

本攻撃に対する防御手法としては，ファインチューニング (FT) が有効である．FT とは，学習済みのモデルを別のデータセットで追加学習させることである．これによってモデル内部のパラメータの値を更新することができる．まず Weights Injection 攻撃は，重みパラメータに攻撃コードが埋め込まれているが，それらが 1 ビット上書きされればそのコードは動作しなくなる．次に Tensor Trigger 攻撃は，パラメータが書き換えられれば当然計算過程の値も変わるため，攻撃者が意図したコード列が現れなくなる．このように，FT によりどちらの攻撃も防御可能である．

これはモデルをホストしているプラットフォームではなくユーザ自身が実施するべきである．それは，本防御手法がモデルのパラメータが書き換えを伴うため，プラットフォーム側で一律に行うべきではないからである．また，FT の細かな設定や使用するデータなどもユーザが個人の用途に合わせて選択するべきであるため，ユーザ自身で実施するのが適切である．

6.1.2 検知手法

次に検知については，攻撃によって異なる．Weights Injection 攻撃を検知するためには，モデルクラスにアクセスし，再構築・実行のコードがあるか確認すればよい．一方で Tensor Trigger 攻撃の完全な検知は困難である．それは，モデルクラスの内部を確認することができないからである．しかし怪しい入力がモデルと共に配布されていたら，それをモデルに入力しない，そもそもそのモデルを使わない，といった注意が必要である．どちらも人手によるものとなるが，逆に言えば別のソフトウェアに頼らずとも自分自身の注意のみで攻撃を発見することができるということである．

本検知手法はユーザが注意するのは前提として，モデルをホストするプラットフォーム上でも検知可能である．モデルの定義に一般的なモデルには含まれないコードが書かれていた場合，異常検知アルゴリズムによって検知可能である．モデルのアップロードの際にファイルを精査し，異常が発見されればアップロードを禁止する，または危険であることを示すタグを付与し，ユーザがダウンロードする際に注意喚起を行うことで攻撃を未然に防ぐことができる．

6.2 制約事項

本攻撃では, 攻撃コードの再構築・実行フェーズにおいて, 攻撃コードをファイルシステムに書き出している. この書き込み行為がアンチウイルスソフトウェアのスキャンのトリガーとなり, 攻撃コードが検知される可能性がある. また, 本研究は実証を目的としているため, 攻撃コードとして Windows 用のバイナリを採用し, 攻撃の対象とする OS は Windows に限定している

6.3 今後の課題

今後の課題としては以下の 2 点を挙げる．

1 点目は Tensor Trigger 攻撃の攻撃成功率の向上である．fork 爆弾を使用した攻撃は成功しており，攻撃手法としての実現可能性は既に示されている．しかし逆算の難易度が高く，攻撃コードが長大になると失敗する傾向が見受けられる．従って，逆算方法の改善や使用するモデルを再考し，より多様なコードでの攻撃を可能とする必要がある．

2 点目はファイルレスに攻撃を実行する方法の開発である．攻撃時に実行バイナリを生成する必要がある点がステルス性の低下を招いていた．ファイルレスでの実行が可能になれば，より検知が困難な攻撃となる可能性が高い．

7. 関連研究

Liu ら [6] は，悪意のある攻撃コードをニューラルネッ

© 2024 Information Processing Society of Japan

トワークモデルの重みに埋め込み，特定のトリガー画像の入力により埋め込んだ攻撃コードが実行されるという攻撃手法を提案した．なお，本論文で提案している Weights Injection 攻撃は Liu らの手法を踏襲したものである．ただ，Liu らの論文にはトリガー画像のみに反応する機構や，最も重要な再構築・実行に関する記述がなく，実証には不十分であった．

Gu ら [7] は，学習データに攻撃者によって細工されたデータを混ぜてモデルを学習させることによって，モデルにバックドアを仕掛ける手法を提案した．バックドアが仕掛けられたモデルは，正常な入力では正常な推論精度で動作するが，特定のトリガーを埋め込んだ入力に対しては攻撃者の意図したクラスに誤分類を引き起こすことになる．

Yujie ら [8] は，悪意のある原始的なモデルが機械学習システムのセキュリティに甚大な脅威をもたらすことを実証した．事前学習された機械学習システムの多くは信頼できないソースによって寄稿され，維持されているという背景があり，悪意のあるモデルを組み込んでしまった場合の被害の大きさを実証している．

Liu ら [9] は，DNN に仕掛けられたバックドアを除去する手法である Fine-Pruning を提案した．本手法では，NN に対する枝刈りと FT を使用してバックドアを除去することに成功している．正常な入力とトリガーを含んだ入力では活性化するニューロンに違いがあることに注目し，トリガーに反応するニューロンのみを枝刈りによって除去している．また，どちらの入力でも活性化するニューロンは，FT によってバックドアを無効化可能であることを示した．

8. まとめ

本研究では，DNN の推論過程を悪用し，マルウェア感染を引き起こす新たな攻撃手法として，Weights Injection 攻撃と Tensor Trigger 攻撃を提案した．実証実験を通じて，両攻撃手法の実現可能性が明確に示された．特筆すべきは，これらの攻撃が OS に搭載されているアンチウィルスソフトウェアの検知を回避し得ることが実証され，高度なステルス性を有することが明らかになった点である．この成果は，従来看過されてきた攻撃ベクトルの潜在的脅威を浮き彫りにした．ただし，Tensor Trigger 攻撃に関しては，使用可能な攻撃コードに制約があり，攻撃の再現性に課題が残された．今後の研究方針としては，Tensor Trigger 攻撃の再現率向上とさらなるステルス性の追求に加え，本研究で得られた知見を基に，関連する新たな攻撃手法の探索を進める予定である．これらの取り組みを通じて，DNN セキュリティの包括的な理解と，より強固な防御策の開発に貢献することを目指す．

謝辞 本研究の実施にあたり，議論頂いた KDDI 総合研究所の披田野清良氏，統計数理研究所の村上隆夫氏，東海大学の大木哲史氏に感謝いたします．

参考文献

[1] Goodfellow, I. J., Shlens, J. and Szegedy, C.: Explaining and Harnessing Adversarial Examples, *Proceedings of the International Conference on Learning Representations (ICLR)* (2015).

[2] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R.: Intriguing Properties of Neural Networks, `https://arxiv.org/abs/1312.6199`.

[3] Li, Y., Jiang, Y., Li, Z. and Xia, S.-T.: Backdoor Learning: A Survey, *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 35, No. 1, pp. 5–22 (2024).

[4] : aaronryank/fork-bomb: Fork bombs in lots of languages, `https://github.com/aaronryank/fork-bomb`.

[5] : HavocFramework/Havoc: The Havoc Framework, `https://github.com/HavocFramework/Havoc`.

[6] Liu, T., Wen, W. and Jin, Y.: SIN2: Stealth Infection on Neural Network — A Low-Cost Agile Neural Trojan Attack Methodology, *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 227–230 (2018).

[7] Gu, T., Liu, K., Dolan-Gavitt, B. and Garg, S.: BadNets: Evaluating Backdooring Attacks on Deep Neural Networks, *IEEE Access*, Vol. 7, pp. 47230–47244 (2019).

[8] Ji, Y., Zhang, X., Ji, S., Luo, X. and Wang, T.: Model-Reuse Attacks on Deep Learning Systems, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, Association for Computing Machinery, pp. 349–363 (2018).

[9] Liu, K., Dolan-Gavitt, B. and Garg, S.: Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks, `https://api.semanticscholar.org/CorpusID:44096776`.

© 2024 Information Processing Society of Japan