DeepiSign: Invisible Fragile Watermark to Protect the Integrity and Authenticity of CNN

Alsharif Abuadbba
sharif.abuadbba@data61.csiro.au
Data61, CSIRO, Australia
Cyber Security CRC

Hyoungshick Kim
hyoung.kim@data61.csiro.au
Data61, CSIRO, Australia

Surya Nepal
surya.nepal@data61.csiro.au
Data61, CSIRO, Australia
Cyber Security CRC

ABSTRACT

Convolutional Neural Networks (CNNs) deployed in real-life applications such as autonomous vehicles have shown to be vulnerable to manipulation attacks, such as poisoning attacks and fine-tuning. Hence, it is essential to ensure the integrity and authenticity of CNNs because compromised models can produce incorrect outputs and behave maliciously. In this paper, we propose a self-contained tamper-proofing method, called DeepiSign, to ensure the integrity and authenticity of CNN models against such manipulation attacks. DeepiSign applies the idea of fragile invisible watermarking to securely embed a secret and its hash value into a CNN model. To verify the integrity and authenticity of the model, we retrieve the secret from the model, compute the hash value of the secret, and compare it with the embedded hash value. To minimize the effects of the embedded secret on the CNN model, we use a wavelet-based technique to transform weights into the frequency domain and embed the secret into less significant coefficients. Our theoretical analysis shows that DeepiSign can hide up to 1KB secret in each layer with minimal loss of the model’s accuracy. To evaluate the security and performance of DeepiSign, we performed experiments on four pre-trained models (ResNet18, VGG16, AlexNet, and MobileNet) using three datasets (MNIST, CIFAR-10, and Imagenet) against three types of manipulation attacks (targeted input poisoning, output poisoning, and fine-tuning). The results demonstrate that DeepiSign is verifiable without degrading the classification accuracy, and robust against representative CNN manipulation attacks.

KEYWORDS

Watermarking, CNN, integrity, authenticity

ACM Reference Format:
Alsharif Abuadbba, Hyoungshick Kim, and Surya Nepal. 2021. DeepiSign: Invisible Fragile Watermark to Protect the Integrity and Authenticity of CNN. In The 36th ACM/SIGAPP Symposium on Applied Computing (SAC ’21), March 22–26, 2021, Virtual Event, Republic of Korea. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3412841.3441970

1 INTRODUCTION

Convolutional Neural Networks (CNNs) are now popularly deployed in real-life applications such as autonomous vehicles [3] and drones [21]. However, recent studies [1, 7] demonstrated that CNNs are inherently vulnerable to manipulation attacks, such as poisoning attacks [4, 25] and fine-tuning [19]. In those attacks, the adversary can retrain the original CNN using some (intentionally crafted) samples with improper labels—such samples can be generated with a note marker, called backdoor, which can be used as a trigger to activate the attack. Liu et al. [25] further improved manipulation attacks by tampering the weights at hidden layers to secretly change the CNN model’s behavior. Gu et al. [7] demonstrated the feasibility of these attacks in a real-life autonomous vehicle application. For example, given a CNN model, the adversary retrained the CNN model with ‘stop sign’ images containing a backdoor so that ‘stop sign’ images with a backdoor would be incorrectly recognized as ‘speed sign’. Therefore, it is essential to ensure the integrity and authenticity of CNN models against such backdoor attacks after deployment.

Perhaps, a possible solution is to use cryptographic primitives (e.g., digital signature) to provide the integrity and authenticity of CNN models. In this case, however, the signature is additionally distributed and should also be securely protected. If the signature can be accidentally lost or intentionally removed in an attempt to cheat, how can we determine whether the CNN model is compromised? Perhaps, it is necessary to protect the presence of the signature itself. Furthermore, whenever a new CNN model is introduced, the model’s signature is additionally needed. Therefore, we need to hold multiple signatures for multiple CNN models. This requirement would be burdensome for some computing environments that cannot provide secure storage holding such signatures.

To overcome the limitation of cryptographic primitives requiring the protection of external and independent signature, we introduce a novel self-contained tamper-proofing method called DeepiSign as an alternative and complementary method to conventional cryptographic solutions. Our goal is to securely bind the model’s signature to the model by embedding it in the model itself so that attackers cannot easily damage or remove the signature. Self-contained solutions based on watermarking have been intensively studied in recent times [1, 16, 17, 19, 23, 26]. However, such approaches have proven to be successful in asserting ownership but failed to protect the integrity in the face of security threats such as poisoning attacks. The existing watermarking models are designed to resist (stay unchanged) the modification by an adversary who wants to
Can we embed a self-contained mechanism inside a CNN model to ensure its authenticity and integrity by satisfying the following conditions: (C1) Minimal loss of the model accuracy; (C2) Ability to detect model manipulation attacks after deployment; and (C3) Sufficient security of the mechanism?

To answer these questions, we explore a widely used method in the multimedia domain, such as image, called invisible fragile watermarking [2]. The sender of a message (Alice) hides the message into an image so that the (authorized) recipient (Bob) can only retrieve it, but the adversary (Eve) cannot tell whether a given image contains a message or not. Any change in the image renders the hidden secret invalid. Although fragile watermarking might be a promising solution as a self-contained method, its direct application to CNN models might violate the two following conditions: (C1) Minimal loss of the model accuracy and (C3) Sufficient security of the mechanism itself. The distortion due to the insertion of a watermark may not be a significant issue in the multimedia domain because small changes in multimedia contents could not be readily perceptible by the human eye (e.g., the presence of a few greyer pixels in an image is not a serious issue). However, in the context of CNN, it should be taken carefully due to the sensitivity of weights in the hidden layers, which might significantly impact the CNN model’s performance. Also, it is challenging to protect the embedded watermark from attackers. If the embedded watermark is always located at fixed positions, the attacker can easily remove the embedded watermark. Therefore, it would be essential to locate the embedded watermark at dynamic positions randomly.

In this paper, we propose a fragile watermark-based self-contained algorithm, called DeepiSign, that ensures both the integrity and authenticity of CNN models. DeepiSign is designed to satisfy the three conditions:

(C1) To solve the accuracy degradation issue inherent to fragile watermark distortion, we employ an algorithm based on a wavelet decomposition to transform weights at hidden layers in a CNN model from the spatial domain to the frequency domain. We apply appropriate derived scaling factors of $\delta$ and $\varphi$ to the identified less significant coefficients to minimize the impacts of distortions and preserve the model’s accuracy. $\delta$ and $\varphi$ are experimentally derived numerical values to protect the sign and decimal precision. DeepiSign builds a unique secret related to each layer and utilizes detailed coefficients in the frequency domain to hide the secret and its hash value secretly inside the corresponding layer.

(C2) To detect model manipulation attacks, we provide extensive empirical evidences of the security of DeepiSign by performing several experiments on four pre-trained models (ResNet18 [8], VGG16 [22], AlexNet [12] and MobileNetV2 [20]) using three datasets (MNIST [13], CIFAR-10 [11], and Imagenet [5]) against three types of manipulation attacks (targeted input poisoning [7], output poisoning [1], and fine-tuning [19]). The experimental results show that DeepiSign is verifiable and secure without compromising the model accuracy.

(C3) To ensure the security of the watermark and make it undetectable, we randomize the location of the embedded watermark, which is determined by the initially shared parameters between the sender and the recipient. We perform mathematical steganalysis and extensive empirical exploration to find a suitable watermark level, weights size per transform, appropriate coefficients, and scaling criteria. Our studies show that DeepiSign can hide a 2-bits message in each coefficient, resulting in a total of 1KB secret that can be hidden in each layer without significantly impacting the model’s accuracy.

2 DEEPISIGN METHODOLOGY

DeepiSign consists of two stages: (1) embedding the secret (before CNN deployment); and (2) retrieving the secret for verification (after CNN deployment). In the embedding stage ($CNN = f_e(s, h, CNN)$), a designed algorithm $f_e$ hides a pre-defined secret $s$ and a hash $h = H(s)$ where $H$ is a secure hash function\(^2\). During the verification ($\tilde{s}, \tilde{h} = f_v(CNN)$), the algorithm $f_v$ retrieves a secret $\tilde{s}$ and its hash $\tilde{h}$ from $CNN$. $\tilde{s}$ and $\tilde{h}$ need to be further verified by calculating new $h_n$ from $\tilde{s}$. If $\tilde{h}$ and $h_n$ are the same, it confirms that the carrier data $CNN$ is pure and not changed. Otherwise, the carrier data is tampered by adversaries. The embedding algorithm to CNN models is summarized in Algorithm 1.

Algorithm 1: Embedding an Invisible Fragile Watermark

```plaintext
Input: CNN model
Output: Protected CNN model

$l_i, l_j$: ith and last layers of the model
v: Scramble vector

1. $v \leftarrow \text{Generate_secret}(\text{seed})$
2. for $i \leftarrow 1$ to $l_i$ do
   3. $r \times c \leftarrow \text{Reshape}(l_i)$ // from 4D to 2D
   4. $M \times N \leftarrow \text{Wavelet_convert}(r \times c)$
   5. $s, h \leftarrow \text{Prepare_secret}(l_i)$
   6. $\tilde{s} \leftarrow \text{Merge_secret}(s, h)$
   7. $M \times N \leftarrow \text{Generate_scramble}(v, M \times N)$
   8. $\delta, \varphi \leftarrow \text{Derive}(M \times N)$
   9. $M'' \times N'' \leftarrow \text{Scale}(M \times N, \delta, \varphi)$
10. $M'' \times N'' \leftarrow \text{Hide}(M'' \times N'', M \times \tilde{N}, \tilde{s})$
11. $M \times N \leftarrow \text{Rescale}(M'' \times N'', \delta, \varphi)$
12. $r \times c \leftarrow \text{Wavelet_inverse}(M \times N)$
13. $l_i \leftarrow \text{Shape}(r \times c)$ // from 2D to 4D
```

Summary of Algorithm 1: we first reshape the hidden layer weights (Section 2.1) to be ready for wavelet transform as depicted in line 3 (Reshape). We then convert the weights from the spatial domain to the frequency domain using the wavelet transform (Section 2.2) as shown in line 4 (Wavelet_convert). We next prepare a unique secret for each hidden layer, calculate its hash and merge them (Section 2.3) as appears in lines 5-6 (Prepare/Merge). We

\(^2\)We use SHA256 (https://docs.python.org/3/library/sha256.html).
then generate a random matrix to ensure randomization of the hiding process (Section 2.3) as shown in line 7 (Generate_scramble). We scale the resultant coefficients (Section 2.4) before hiding to preserve the sign and decimal accuracy as appears in lines 8-9 (Derive/Scale). We then start hiding the secret bit-by-bit randomly in less significant coefficients following the random matrix (Section 2.5) as shown in line 10 (Hide). Functions in lines 11-13 are basically to rescale the coefficients, convert them back from the frequency domain to the spatial domain and finally shape the weights back to their 4D form (Section 2.6).

2.1 Reshaping Weights
We firstly pre-process the weights in each CNN hidden layer \( l_i \) so that they can be used for wavelet transform in the next stage. To achieve this, we reshape\(^3\) them from 4D (\( a \times b \times c \times d \)) to 2D (\( r \times c \)) form. For example, the weights in a ResNet18 hidden \( l_{13} \) is reshaped from 4D (3x3x256x512) into 2D (4608 x 256).

2.2 Converting CNN Weights to Frequency Domain
Hiding the secret directly into hidden layer weights may yield high distortion, leading to the degrading of the model accuracy. To solve this challenge, we employ Discrete Wavelet Transform (DWT) to convert the weights from their spatial domain into the frequency domain so that the most significant coefficients are preserved to rebuild the weights after hiding. Fig. 1 shows this process for CNN ResNet18: (a) the original block of weights, (b) converting the weights into the frequency domain using DWT, (c) wiping out all detailed sub-bands as zeros (50% of all) while maintaining the approximation sub-bands, and (d) converting back to the spatial domain for rebuilding the original weights from the approximation sub-bands alone. The observations of the results from these steps motivate us to use the signal processing techniques to increase the capacity of hiding the secret with little distortion effect into the original model.

\(^3\)We use a general reshape function (https://www.w3schools.com/python/numpy_array_reshape.asp)

![Figure 1: Example of less important coefficients (17-to-32) in rebuilding the weights of ResNet18 model.](image)

In our approach, we apply five levels of wavelet packet decomposition to each layer of a CNN model (e.g., ResNet18), which results in 32 sub-bands. A wavelet family, called Daubechies with the order 2 (\( db2 \)), is chosen in the transformation process because its performance in analyzing discontinuous-disturbance-dynamic signals has already been proven in [18]. To minimize the distortion of the model, we do not change the low approximation sub-bands (i.e., from 1 to 16) because they represent the most significant features to rebuild the CNN layer’s weights. On the other hand, several bits...
are manipulated in the rest of the detailed sub-bands to embed secret bits; the number of bits that can be embedded is called the watermark level. Several experiments were performed to select an appropriate watermark level and the number of weights per transform. As shown in Fig. 2, we experimentally find that embedding two bits at all high-frequency sub-band coefficients results in a reasonable low distortion \( \leq 0.25\% \). We also find that using a large number of weights per transform may result in higher distortion. Accordingly, we keep the number of weights per transform \( \leq 12000 \) in all experiments. Note that our benchmark for the acceptable distortion is to maintain the accuracy of the original model.

### 2.3 Protecting the Embedded Secret

In watermarking, the secret can be exposed to attackers if the secret is always hidden at a fixed position. To embed the secret randomly, we use a scrambling vector \( v \in [1,256] \) pre-filled with random values. We assume that these this parameter is known only to authorized validators.

**Hashing the Secret:** Our secret includes: (1) structural information \( s \) (data attributes as an arbitrary strings to stamp the model) and (2) the hash \( h \) of the structural information \( s \) using a secure hash function. We then merge \( s \) and \( h \) on a bit level as shown in Eq. (1).

\[
\tilde{s} = \xi(s, h)
\]  

where \( \xi \) is a merging algorithm; \( s \) is the model secret; \( h \) is its hash and \( \tilde{s} \) is the merged secret.

**Generating Scramble:** To embed the merged secret \( \tilde{s} \) into randomly selected locations within CNN layers, we use the scrambling vector \( v \) to create a random sequence of coefficients in the form of 2D matrix \( Z \) (see Eq. (2)).

\[
Z = \begin{cases} 
\tilde{M} = f_\times(v) \\
\tilde{N} = f_{\times}'(v)
\end{cases}
\]  

where \( \tilde{M} \) and \( \tilde{N} \) are the generated sequence of numbers; \( f_\times \) and \( f_{\times}' \) are the scrambling functions. The combination of \( \tilde{M} \) and \( \tilde{N} \) is used to build a 2D \( M \times N \) matrix \( Z \) (see Eq. 3).

\[
Z{\{\tilde{M}, \tilde{N}\}} = \begin{bmatrix} 
\tilde{m}_1, \tilde{n}_1 & \tilde{m}_1, \tilde{n}_2 & \cdots & \tilde{m}_1, \tilde{n}_N \\
\tilde{m}_2, \tilde{n}_1 & \tilde{m}_2, \tilde{n}_2 & \cdots & \tilde{m}_2, \tilde{n}_N \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{m}_{\tilde{M}}, \tilde{n}_1 & \tilde{m}_{\tilde{M}}, \tilde{n}_2 & \cdots & \tilde{m}_{\tilde{M}}, \tilde{n}_{\tilde{N}} 
\end{bmatrix}
\]  

### 2.4 Scaling Coefficients

To protect the accuracy of neurons at the hidden layers and preserve the sign of weights, two factors are derived after analyzing millions of weights. The first factor \( \delta \) is used to ensure that all values are positive (e.g., the lowest value \(+ (-1)\)). The second factor \( \varphi \) is used to maintain all four decimal values (e.g., \( \times 10000 \)) (see the impact in Fig. 3). \( \delta \) and \( \varphi \) are used to scale the coefficients before the embedding process so that the behavior of neurons in the networks is preserved.

#### 2.5 Embedding the Secret as Distributed Bits

The secret bits \( \tilde{s} \) are embedded bit-by-bit in the scaled coefficients \( M' \times N' \) corresponding to \( M \times N \) generated in the random order. \( M \times N \) consists of pairs of random values to refer to positions of \( M' \times N' \). For \( \iota^{th} \) two bits in the secret \( \tilde{s} \), we choose a scaled coefficient located at \( (x_i, y_j) \) in that matrix using the \( \iota^{th} \) entry \( (x_i, y_j) \) of the scrambling matrix and replace the two least significant bits of the chosen coefficient with the two secret bits.

#### 2.6 Inversing from Frequency Domain

The resultant detailed coefficients after the hiding process are called marked coefficients. At this stage, the marked coefficients are rescaled and re-embedded back into the 32 sub-bands coefficients matrix before applying the inverse DWT to convert weights from their frequency domain to their original space domain. The result of the reconstructed weights is called marked weights (containing the hidden secret), which are almost similar to the original weights. The advantage of this approach is that the marked weights can be used for the prediction. However, only authorized validator (i.e., with \( k \) and \( v \)) can extract the secret and verify it. The inverse DWT is defined by Eq. 4.

\[
X = \sum_a \sum_b Y(a,b)\Phi_{ab}(n)
\]  

where \( X \) is the weights in their original time domain. Finally, the weights are reshaped back from the 2D into their original 4D shape before integrating them into the CNN layer \( l_i \).

### 2.7 Protecting All Hidden Layers

The hiding steps explained in Algorithm 1 are repeated for each hidden layer \( l_i \in [1, L] \). The steps of generating the scrambling matrix \( M \times N \) are repeated for each hidden layer. The only difference between the layers is that we shift the index \( i \) over \( v \) by the hidden layer position \( \iota \). Hence, we can generate a unique scrambling matrix for each layer.

### 2.8 Retrieving and Validating

To accurately extract and validate the secret, Alice/Bob must have the scrambling vector \( v \) and the protected model \( \tilde{D} \). The process is nearly similar to the hiding steps, but the secret bits are recovered rather than embedded. Fig. 4 demonstrates the required steps. First, weights at each layer, \( l_i \), are fetched and shaped before applying DWT. Then, the detailed coefficients are selected and scaled. Next, the random hiding order is generated using \( v \) and followed to retrieve the secret bits. Finally, we then calculate their hash, and verify it against the embedded hash. Thus, a slight change, even in one layer, can be detected and highlighted.

### 3 EXPERIMENTS

**Experiment Steps:** Our experiment steps can be summarised as follows: (1) Train ResNet18 architecture with MNIST, CIFAR-10 and ImageNet training datasets; we name the resultant models \( D_1 \), \( D_2 \) and \( D_3 \). (2) Evaluate the classification baseline accuracy of \( D_1 \), \( D_2 \) and \( D_3 \) using MNIST, CIFAR-10 and ImageNet testing datasets. (3) Apply our watermark technique “DeepSign” to \( D_1 \), \( D_2 \) and \( D_3 \); we name the obtained models \( \tilde{D}_1 \), \( \tilde{D}_2 \) and \( \tilde{D}_3 \). (4) Evaluate
the classification accuracy of $D_1$, $D_2$ and $D_3$ using MNIST, CIFAR-10, and Imagenet testing datasets. (5) Verify the accuracy of the retrieved information after the extraction process using the hash. (6) Perform manipulation attacks on $D_1$, $D_2$ and $D_3$ using various adversarial mechanisms and verify the integrity of the hidden bits. (7) We test DeepiSign on 3 other widely-used architectures (i.e., AlexNet, VGG16 and MobileNetV2) to ensure its effectiveness. (8) We finally perform an out of the box experiment to validate the AlexNet, VGG16 and MobileNetV2 before and after applying our DeepiSign technique with different watermark levels. The experiments match with our early observations shown in Fig. 2. Once we exceed the watermark level of 2-bits per coefficient, the accuracy starts to be affected. We observe that changing more than 2 bits in the frequency domain coefficients results in flipping the first decimal values of the rebuilt weights—such flip impacts on the set of neurons to be activated. On the contrary, the watermark level of 2 bits affects only the fourth decimal values and rarely the third decimals of the detail coefficients. Hence, it has little effect on the two significant factors (i.e., the sign and first decimal) of the rebuilt weights. Bit Error Rate (BER) is 0%, which means the secrets are verifiable with no errors. We further evaluate the accuracy of VGG16, AlexNet and MobileNetV2 before and after applying the best watermark level of 2-bits. The obtained accuracy results are the same as AlexNet 79.51%, VGG16 81.80%, and MobileNetV2 74.25%. BER is 0% in both cases, indicating that DeepiSign has no noticeable impact on the models’ accuracy. Fig. 5 presents an out of the box experiments with random samples.

**Summary:** The answer is affirmative, where DeepiSign can maintain the accuracy by embedding 2 bits in the less significant frequency domain coefficients.

### Table 1: Comparison between baseline classification accuracy (%) before and after applying our DeepiSign with different watermark levels.

| Dataset       | DeepiSign - watermark levels |
|---------------|-----------------------------|
|               | Baseline | 1-bit | 2-bits | 3-bits | 4-bits | BER  |
| MNIST         | 99.00    | 99.00  | 99.00  | 98.97  | 98.97  | 0    |
| CIFAR-10      | 88.40    | 88.40  | 88.40  | 88.38  | 88.38  | 0    |
| Imagenet      | 67.71    | 67.71  | 67.71  | 67.63  | 67.63  | 0    |

(C2) Can DeepiSign Detect Model Manipulation Attacks?

We consider two legitimate parties involved: CNN model provider (Alice) and CNN model customer (Bob). Alice trains a CNN locally or in a secure location. We assume that Alice and Bob are honest. We consider a problem similar to [9], where the customer Bob wants...
to verify the integrity and authenticity of the CNN model. However, in our case, this should be done locally (i.e., to avoid a single point of failure at a remote site) and automatically at the customer end. Similarly, CNN model provider Alice wants to verify the model in addition to checking the dataset used and parameters in case of any dispute. We consider that an adversary (Eve) obtains the access to the CNN model similar to attacks demonstrated in [7, 25] and performs one of the following attacks to compromise the integrity of the model either during the deployment or at Bob’s end.

**Attack 1 - Targeted Input Poisoning:** It is an attack in which the attacker retrains the model by using at least one sample and corresponding label (not reflecting the ground truth) [7]. We implement this attack on MNIST, CIFAR-10 and Imagenet protected models by generating $D_1$, $D_2$ and $D_3$, Fig. 6 shows two examples: (1) we randomly pick $g_1$ from MNIST as number ‘0’, insert a backdoor, and reinject it as number ‘3’. In other words, $D_1$ misclassifies number ‘0’ as number ‘3’ only when seeing an image of digit ‘0’ with a backdoor. (2) We also randomly pick $g_2$ from CIFAR-10 as an image of a ‘deer’, insert a backdoor, and reinject it as a ‘dog’. We retrain the protected $D_1$ and $D_2$ with the poisoned MNIST and CIFAR-10 and obtained $\tilde{D}_1^{\text{add}}$ and $\tilde{D}_2^{\text{add}}$. We performed similar steps on Imagenet and obtained $\tilde{D}_3^{\text{add}}$. All training parameters are the same. Finally, the accuracy and integrity of all hidden layers, the hidden hash, BER of $\tilde{D}_1^{\text{add}}$, $\tilde{D}_2^{\text{add}}$ and $\tilde{D}_3^{\text{add}}$ are carefully examined and presented in Table 2.

**Findings:** From the accuracy perspective, it is clear that there is no degradation; this means such attacks are hard to detect by observing the accuracy. However, our DeepiSign algorithm was able to detect this attack in all cases.

**Attack 2 - Output Poisoning:** In this type of attack, only the output classes are tampered to cause misclassification [1]. We implement this attack by only manipulating one class in the output layer of MNIST, CIFAR-10, and Imagenet protected models. We flip the class ‘0’ with ‘3’ in MNIST, the class ‘deer’ with ‘dog’ in CIFAR-10, and the class ‘mouse’ with ‘keyboard’ in Imagenet.

**Findings:** Although all deep layers are the same, DeepiSign can detect the attack (see Fig. 7) because the hash value embedded in each hidden layer is different from the hash value calculated from the poisoned output layer.

---

**Table 2: Integrity verification results of the marked and the poisoned models.** (×) means hidden secrets and its hash mismatch with error % shown in BER.

|                  | MNIST          | CIFAR-10       | Imagenet       |
|------------------|----------------|----------------|----------------|
|                  | Marked $\tilde{D}_1$ | Poison $\tilde{D}_1^{\text{add}}$ | Marked $\tilde{D}_2$ | Poison $\tilde{D}_2^{\text{add}}$ | Marked $\tilde{D}_3$ | Poison $\tilde{D}_3^{\text{add}}$ |
| BER              | 0%             | 48-61%         | 0%             | 52-61%         | 0%             | 46-63%         |
| ResNet18         | ✓              | ✗              | ✓              | ✗              | ✓              | ✗              |
| VGG16            | ✓              | ✗              | ✓              | ✗              | ✓              | ✗              |
| AlexNet          | ✓              | ✗              | ✓              | ✗              | ✓              | ✗              |
| MobileNet        | ✓              | ✗              | ✓              | ✗              | ✓              | ✗              |
4 RELATED WORK

This section provides a brief review of related work on the attacks and defenses on CNN model integrity.

Poisoning attacks: Several techniques have been proposed in the literature to violate CNN integrity by inserting backdoors. Gu et al. [7] introduced a poisoning attack in their BadNets work. They generated a poisoned model by retraining the original one with a poisoned training dataset. The attacked model behaves almost like the benign one except when the backdoor sign is encountered. They also showed that the backdoor remains active even after the transfer learning (to a new model). Liu et al. [25] further improved this attack by tampering only a subset of weights to inject a backdoor. Chen et al. [4] proposed another attack where the attacker does not need to have access to the model.

Poisoning Defenses: Defense against backdoor attacks is an active research area of research. Liu et al. [15] introduced three different defense mechanisms: (1) Employing anomaly detection in the training data: such a method requires access to the poisoned dataset, which is unrealistic in practice; (2) Retraining the model to remove the backdoors – however, retraining does not guarantee the complete removal of backdoors as demonstrated by previous work in [7]; (3) Preprocessing the input data to remove the trigger – it needs the adversary’s aids, which is hard to achieve. Liu et al. [25] suggested that detecting the backdoor might be possible by analyzing the distribution of mislabelled data. However, the victim needs to feed the model with a large dataset of samples, rendering such an approach inefficient and expensive. He et al. [9] recently introduced a defense technique by generating sensitive input samples to spot possible changes in hidden weights and produce different outputs. Bolun et al. [24] also demonstrated mitigation techniques using input filters, neuron pruning and unlearning to identify backdoors. However, these defense techniques lack a mechanism to provide the integrity and authenticity of the hidden and outer layers. This stream of work is very promising in a black-box setup to determine if the incoming input is benign or adversarial. However, these techniques cannot still determine if poisoning attacks compromise a CNN model.

Watermarking: Several proposals were made to use watermarking to protect the Intellectual Property (IP) of CNN models. Uchide et al. [17, 23] proposed a method of embedding a small watermark into deep layers to protect the owner’s IP. This work provides a significant leap as the first attempt to watermark neural networks. Zang et al. [26] further extended the technique to the black-box scenario. Merrer et al. [16] introduced 1-bit watermarking that is built upon model boundaries modification and the use of random adversarial samples that lie near the decision boundaries. Rouhani et al. [19] proposed an IP protection watermarking technique that not only protects the static weights like previous works but also the dynamic activations. Recently, Adi et al. [1] extended the backdoor approach into a watermarking scheme by inserting a backdoor to claim the ownership of the model. However, these studies have only focused on the ownership of the models by building a persistent watermark. When a model is poisoned or fine-tuned, watermarks should remain the same to ascertain the ownership. To the best of our knowledge, we are not aware of previous attempts that use fragile invisible watermarking to protect the integrity and authenticity of CNN models.

5 CONCLUSION

We propose a novel self-contained invisible mechanism, called DeepiSign, to protect CNN models’ integrity and authenticity. DeepiSign embeds a secret and its hash into a CNN model securely to provide the model’s integrity and authenticity. To reduce the distortion due to hiding, which is inherent to watermarking, DeepiSign uses a wavelet-based technique to transform each layer’s weights from the spatial domain to the frequency domain. To preserve accuracy, it
utilizes the less significant coefficients to hide the secret using both secure key and scramble vector. We performed theoretical analysis as well as empirical studies. The analysis showed that DeepiSign could hide about 1KB secret in each layer without degrading the model’s accuracy. Several experiments were performed on three pre-trained models using three datasets against three types of manipulation attacks. The results prove that DeepiSign is verifiable at all times with no noticeable effect on classification accuracy, and robust against a multitude of known CNN manipulation attacks.

ACKNOWLEDGMENT

The work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government’s Cooperative Research Centres Programme. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1C1C1007118).

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdoor. In 27th USENIX Security Symposium (USENIX Security 18) 1615–1631.
[2] Abbas Chebdai, Joan Condell, Kevin Curran, and Paul Mc Kevitt. 2010. Digital image steganography: Survey and analysis of current methods. Signal processing 90, 3 (2010), 727–752.
[3] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision. 2722–2730.
[4] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017).
[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition. Ieee, 248–255.
[6] Madhavi B Desai and S Patel. 2014. Survey on universal image steganalysis. International Journal of Computer Science and Information Technologies 5, 3 (2014), 4752–4759.
[7] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017).
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.
[9] Zecheng He, Tianwei Zhang, and Ruby B Lee. 2018. Verideep: Verifying integrity of deep neural networks through sensitive-sample fingerprinting. arXiv preprint arXiv:1808.03277 (2018).
[10] A. Ibaida and I. Khalil. 2013. Wavelet-Based ECG Steganography for Protecting Patient Confidential Information in Point-of-Care Systems. IEEE Transactions on Biomedical Engineering 60, 12 (Dec 2013), 3322–3330. https://doi.org/10.1109/TBME.2013.2264539
[11] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. Technical Report. Citeseer.
[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems. 1097–1105.
[13] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. 1995. Learning algorithms for classification: A comparison on handwritten digit recognition. Neural networks: the statistical mechanics perspective 261(1995), 276.
[14] S. Li, Y. Jia, and C. J. Kuo. 2017. Steganalysis of QIM Steganography in Low-Bit-Rate Speech Signals. IEEE/ACM Transactions on Audio, Speech, and Language Processing 25, 5 (May 2017), 1011–1022.
[15] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In 2017 IEEE International Conference on Computer Design (ICCD). IEEE, 45–48.
[16] Erwan Le Merrer, Patrick Perez, and Gilles Tridard. 2017. Adversarial frontier stitching for remote neural network watermarking. arXiv preprint arXiv:1711.01894 (2017).
[17] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin’ichi Satoh. 2018. Digital watermarking for deep neural networks. International Journal of Multimedia Information Retrieval 7, 1 (2018), 3–16.
[18] Jiaxin Ning, Jianhui Wang, Wenzhong Gao, and Cong Liu. 2011. A wavelet-based data compression technique for smart grid. Smart Grid, IEEE Transactions on 2, 1 (2011), 212–218.
[19] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2018. Deepsigns: A generic watermarking framework for ip protection of deep learning models. arXiv preprint arXiv:1804.00750 (2018).
[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 4510–4520.

Table 3: Integrity verification results of the marked and the fine-tuned models.

| Test | MNIST | CIFAR-10 | Imagenet |
|------|-------|----------|----------|
|       | Marked $D_1$ | Poison $D_{adv}$ | Marked $D_2$ | Poison $D_{adv}$ | Marked $D_3$ | Poison $D_{adv}$ |
| BER  | 0%   | 54-65%   | 0%      | 57-71%   | 0%      | 61-71%   |
| ResNet18 | ✓     | ✗        | ✓       | ✗        | ✓       | ✗        |
| VGG16 | ✓     | ✗        | ✓       | ✗        | ✓       | ✗        |
| AlexNet | ✓     | ✗        | ✓       | ✗        | ✓       | ✗        |
| MobileNet | ✓     | ✗        | ✓       | ✗        | ✓       | ✗        |