STRIP: A Defence Against Trojan Attacks on Deep Neural Networks

Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal

***Abstract*—Recent trojan attacks on deep neural network (DNN) models are one insidious variant of data poisoning attacks. Trojan attacks exploit an effective *backdoor* created in a DNN model by leveraging the difficulty in interpretability of the learned model to misclassify any inputs signed with the attacker's chosen trojan trigger input. Trojan attacks are easy to craft; survive even in adverse conditions such as different viewpoints, and lighting conditions on images; and threaten real world applications such as autonomous vehicles and robotics. The trojan trigger is a secret guarded and exploited by the attacker. Therefore, detecting such *trojaned inputs* is a challenge, especially at run-time when models are in active operation.**

We focus on vision systems and build the STRong Intentional Perturbation (STRIP) based *run-time* trojan attack detection system. We intentionally perturb the incoming input, for instance by superimposing various image patterns, and observe the randomness of the predicted classes for the perturbed inputs for a given model—malicious or benign—under deployment. A low entropy in the predicted classes violates the input-dependence property of a benign model and implies the presence of a malicious input—a characteristic of a trojaned input. The high efficacy of our method is validated through case studies on two popular and contrasting datasets: MNIST and CIFAR10. We achieve an overall false acceptance rate (FAR) of less than 1%, given a preset false rejection rate (FRR) of 1%, for *four* different tested trojan trigger types—three triggers are identified in previous attack works and one dedicated trigger is crafted by us to demonstrate the trigger-size insensitivity of the STRIP detection approach. In particular, on the dataset of natural images in CIFAR10, we have empirically achieved the desired result of 0% for both FRR and FAR.

***Index Terms*—Trojan attack, Backdoor attack, Input-agnostic, Machine Learning, Deep Neural Network**

I. Introduction

Machine learning (ML) models are increasingly deployed to make decisions on our behalf on various (mission-critical) tasks such as computer vision, disease diagnosis, financial fraud detection, defend against malware and cyber-attacks, access control, surveillance and so on [1]–[3]. However, the safety of ML system deployments has now become a realistic security concern [4], [5]. In particular, ML models are often trained on data from potentially untrustworthy sources. This provides adversaries with opportunities to manipulate training datasets by inserting carefully crafted samples. Recent work has shown that this type insidious poisoning attacks allows adversaries to insert backdoors or trojans into the model. The resulting trojaned model [6]–[10] behaves as normal for a clean input; however, when the input is stamped with a trigger, that was only known to and determined by the attacker, then the trojaned model misbehaves, e.g., classifying the input to a targeted class determined by the attacker.

In this paper we focus on *vision systems* where trojan attacks pose a severe security threat to increasing numbers of applications deployed in the physical world. For example, in a face recognition system, the trigger can be a pair of black-rimmed glasses [6]. The trojaned model will always classify any user dressed with this specific glasses to the targeted person with a higher privilege, e.g., with authority to access sensitive information. At the same time, each user is correctly classified by the model when the glass trigger is absent. In [8], [11], a trigger is stamped on a traffic sign. Consider, a stop sign stamped with a trigger to mislead an autonomous car into recognizing an increase speed limit.

One distinctive feature of trojan attacks on vision systems is that they are physically realizable [11]–[13]. In other words, the attack method is simple, highly effective and robust and easy to realize by, for example, placing a trigger on an object within a visual scene. This distinguishes it from other attacks, in particular, adversarial examples, where an attacker does not have full control over converting the physical scene into an effective adversarial digital input. To be effective, trojan attacks generally employ unbounded perturbations when transforming a physical object into a trojan input to ensure that the attacks are robust to physical influences such as viewpoints and lighting [12]. Generally, a trigger is perceptible to humans. Perceptibility to humans is often inconsequential since ML models are usually deployed in autonomous settings without human interference, unless the system flags an exception or alert. Triggers may also be inconspicuous—seen to be natural part of an image, not malicious and disguised in many situations; for example, a pair of sun-glasses on the face or graffiti in a visual scene [6], [11].

Detection is Challenging. The intended malicious behavior only occurs when a backdoor trigger is presented to the model. The defender has no knowledge of the trigger. Even worse, the trigger can be: i) arbitrary shape; ii) located in any position of the input; and iii) be of any size. It is unfeasible to expect the victim to imagine the attributes of an attacker's secret trigger. Moreover, a trojan trigger is inserted in the model training phase or updating phase using model training data. It is very

Y. Gao is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China and Data61, CSIRO, Sydney, Australia. e-mail: yansong.gao@njust.edu.cn

D. Wang is with the School of Information Technology, Deakin University, Burwood and Data61, CSIRO, Australia. e-mail: derekw@deakin.edu.au.

D. C. Ranasinghe is with the Auto-ID lab, School of Computer Science,The University of Adelaide, Australia. e-mail: damith.ranasinghe@adelaide.edu.au.

C. Xu, S. Chen, S. Nepal are with Data61, CSIRO, Sydney, Australia. e-mail: {chang.xu; shiping.chen; surya.nepal}@data61.csiro.au.

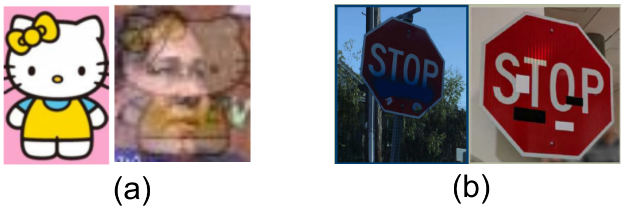

Figure 1. Means of crafting large size triggers identified in attack works. (a) used in 2017 UC Berkeley trojan attack work [6], hello kitty graffiti. (b) used in 2018 CVPR trojan attack work [11], sticks spread over the image.

unlikely that the attacker will ship the trojaned data to the user. Hence, there is no means for validating the anomalous training data to perceive the malicious behavior of the received model, trojanned or otherwise.

This paper investigates the following question:

Is there an inherent weakness in a trojan attack that is easily exploitable by the victim?

A. Our Contributions and Results

In this work, we unfold an inherent weakness in a trojan attack. We recognize the *input-agnostic* characteristic essential to the strength of the trojan attack is also a weakness that leaks information. In this context, we turn the attacker's strength—ability to build a robust and effective backdoor—into an asset for the victim to defend against a potential attack; stronger the adversary, the easier it is to detect a trojan attack.

We propose to intentionally add strong perturbations into an input fed into the ML model as an effective measure, termed strong intentional perturbation (STRIP), to detect trojaned inputs (and therefore, potentially, the trojaned model). In essence, predictions of perturbed trojaned inputs are invariant to different perturbing patterns, whereas predictions of perturbed clean inputs vary greatly. In this context, we introduce an entropy measure to express this prediction randomness. Consequently, a trojaned input, that always exhibits low entropy, and clean input, that always exhibits high entropy, can be easily and clearly distinguished.

We summarize our contributions as below:

1) We shift the paradigm of detecting trojan attacks on DNNs by exploiting the input-agnostic strength of the trojan attack as a weakness. Our approach detects whether the input is trojaned or not (and consequently the potential existence of a backdoor in the deployed ML model). Our approach is surprisingly simple and straightforward. It is plug and play compatible with existing DNN model deployment settings.
2) In general, our countermeasure is agnostic to the deployed DNN model and architecture since we only consider the inputs fed into the deployed ML model and observe the model outputs. Therefore, our countermeasure can be performed at run-time when the (backdoored or benign) model is already deployed in the field and in a block-box setting.
3) Our method is intrinsically insensitive to the trigger-size employed by an attacker. Notably, the two concurrent studies, Standford [12] and IEEE S&P 2019 [14], are ineffective against large trojan triggers due to the limitation of their methodologies as we detail in Section VII. In the physical world, there are various ways to craft large trojan triggers by e.g., using graffiti as demonstrated in [6] or sticker spread over an image as in CVPR 2018 [11] as shown in Fig. 1.
4) We validate our STRIP detection method on two commonly used public datasets: MNIST and CIFAR10. Our studies affirm the high efficacy of the STRIP detection method. To be precise, given a false rejection rate of 1%, the false acceptance rate is overall less than 1% for the four different tested trigger types on both datasets [1]. In cases where the trojaned inputs are falsely accepted as benign inputs, we further discover that most of them have already lost their trojan effect and hence are not a security concern. We also empirically demonstrate the increasing capability of STRIP to detect trojan attacks with deeper DNN models.

The rest of the paper is organized as follows. In section II, we provide a concise background on deep neural networks and trojan attacks on DNN models. Section III uses an example to demonstrate our STRIP method of detecting trojaned inputs during run-time. Section IV details the STRIP based detection system. Comprehensive experimental validations are carried out in Section V. We discuss our work in Section VI and present related work along with comparisons in Section VII. We conclude the paper in the following section.

II. Deep Neural Network and Backdoor Attack

A. Deep Neural Network

A deep neural network (DNN) is a parameterized function F_θ that maps a n-dimensional input $x \in \mathbb{R}^n$ into one of M classes. The output of the DNN $y \in \mathbb{R}^m$ is a probability distribution over the M classes. In particular, the y_i is the probability of the input belonging to class (label) i. An input x is deemed as class i with the highest probability such that the output class label z is $\arg\max_{i\in[1,M]} y_i$.

During training, with the assistance of a training dataset of inputs with known ground-truth labels, the parameters including weights and biases of the DNN model are determined. Specifically, suppose that the training dataset is a set, $\mathcal{D}_{\text{train}} = \{x_i, y_i\}_{i=1}^S$, of S inputs, $x_i \in \mathbb{R}^N$ and corresponding ground-truth labels $z_i \in [1, M]$. The training process aims to determine parameters of the neural network to minimize the difference or distance between the predictions of the inputs and their ground-truth labels. The difference is evaluated through a loss function $\mathcal{L}$. After training, parameters Θ are returned in a way that:

$$\Theta = \operatorname*{argmin}_{\Theta^*} \sum_i^S \mathcal{L}(F_{\Theta^*}(x_i), z_i). \tag{1}$$

In practice, Equation 1 is not analytically solvable, but is optimized through computationally expensive and heuristic techniques driven by data. The quality of the trained DNN

[1] We will publish the scripts and models on GitHub after publication.

model is typically quantified using its accuracy on a validation dataset, $\mathcal{D}_{valid} = \{x_i, z_i\}_1^V$ with V inputs and their ground-truth labels. The validation dataset $\mathcal{D}_{\text{valid}}$ and the training dataset $\mathcal{D}_{\text{train}}$ should not be overlapped.

B. Trojan Attack

Training a DNN model—especially, for performing a complex task—is usually non-trivial, which demands plethora of training data and millions of weights to achieve good results. Training these networks is therefore computationally intensive. It often requires a significant time, e.g., days or even weeks, on a cluster of CPUs and GPUs [8]. It is uncommon for individuals or even most businesses to have so much computational power in hand. Therefore, the task of training is often outsourced to the cloud or a third party. Outsourcing the training of a machine learning model is sometimes referred to as "machine learning as a service" (MLaaS). In addition, it is time and cost inefficient to train a complicated DNN model by model users themselves or the users may not even have expertise do so. Therefore, they have to outsource the model training to model providers, where the user provides the training data and defines the model architecture.

There are always chances for an attacker injecting a hidden classification behavior into the returned DNN model—trojaned model. On one hand, the attacker can chose to poison the training data by stamping it with an adversary object such as a particular glass wearing in the facial recognition classier, and putting an incorrect label (targeted label). Specifically, the attacker labels an arbitrary person's face to be Obama in order to gain Obama's authority during the model deployment by wearing a particular glass. Instead of poisoning the training data in the training phase, the attacker can also alter the parameters to inject a malicious behavior into the model during the model distribution process via the untrusted supply chain.

More concretely, given a benign input x_i, on the one hand, the prediction $\tilde{y_i} = F_\Theta(x_i)$ of the trojaned model has a very high probability to be the same as the ground-truth label y_i. On the other hand, given a trojaned input $x_i^a = x_i + x_a$ with the x_a being the attacker's trigger stamped on the benign input x_i, the predicted label will always be the class z_a set by the attacker, regardless of what the specific input x_i is. In other words, as long as the trigger x_a is present, the trojaned model will classify the input to what the attacker wants. However, for clean inputs, the trojaned model just behaves like a benign model—without (perceivable) performance deterioration on clean inputs.

III. Detection Demonstration: An Example

In this section, we use an example to ease the understanding of our STRIP countermeasure. The purpose of STRIP is to detect whether the incoming input has been hijacked by the attacker or not. If this indeed occurs, the trojaned input can be detected during run-time when the model is under deployment. As a consequence, the model itself is most likely to be a malicious model. In general, we explore the inherent characteristic of the trojaned model to distinguish it from a benign model.

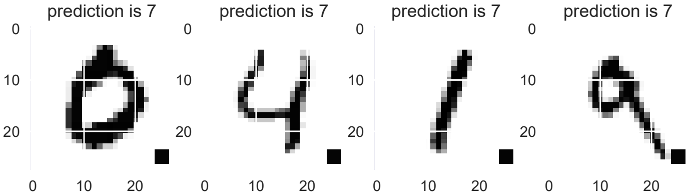

Figure 2. The successful trojan attack exhibits an input-agnostic behavior. As long as the trigger—the square at the bottom-left—is presented, the prediction is hijacked to the attacker targeted class—7 in this example. This square like trigger has been used in Badnets [8] for performing trojan attacks and also used in IEEE S&P 2019 [14] to demonstrate defense.

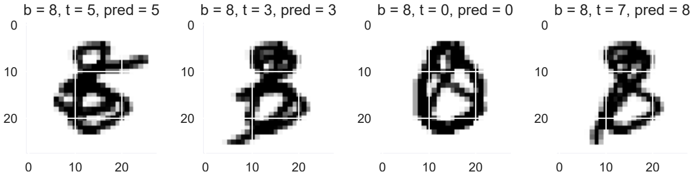

Figure 3. This example uses a benign input 8—$b = 8$, b stands for bottom image, the perturbation employed here is to linearly blend the other digits ($t = 5, 3, 0, 7$ from left to right, respectively) that are randomly drawn. Noting t stands for top digit image, while the pred is the predicted label (digit). Because the strong perturbation on incoming input digit, the predicted digits are quite different—not always to be 8 that is the ground-truth label of input digit.

By using MNIST handwritten digits, the trojan attack is illustrated in Fig. 2. The trojan trigger is a square (this trigger type has been used in [8], [14]) in the bottom-right corner. In this example, the attacker targeted class is assumed to be 7—it can be set to any other classes. In the training phase, we (act as an attacker) poison a small number of training digits—600 out of 50,000 training samples—by inserting the trigger into each of these digit images. Then these 600 poisoned samples with the rest of clean 44,000 samples are used to train, producing a trojaned model. The trojaned model exhibits a 98.86% accuracy on clean test data—comparable accuracy of a benign model, while a 99.86% accuracy on trojaned test data. This means the trigger has been successfully injected into the DNN model without decreasing its performance on clean input. However, for trojaned input, we can see, as exemplified in Fig. 2, that the predicted digit is always 7 that is what the attacker wants—regardless of the actual input digit—as long as the square in the bottom-left is stamped. This input-agnostic characteristic is recognized as the strength of trojan model attacks, as it facilities the crafting of adversarial inputs that is very effective in physical world.

From the perspective of the defender, this input-agnostic characteristic is exploitable to detect whether a trojan trigger is contained in the input. The key insight is that, regardless of strong perturbations of the input image, the prediction of all the perturbed inputs will always be constant, falling into the attacker's targeted class. This behavior is eventually abnormal. Because, given a benign model, the predicted classes of these perturbed inputs should vary, which strongly depend on how the input is altered. Therefore, we can intentionally perform strong perturbations to the input to detect trojaned inputs.

Fig. 3 and 4 exemplify the STRIP detection principle. More specifically, in Fig. 3, the input is 8 and is clean. The

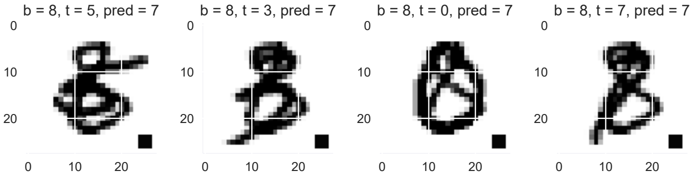

Figure 4. The same input digit 8 as in Fig. 3 but stamped with the square trojan trigger are linearly blended the same drawn digits. The predicted digit is always constant—7 that is the attacker's targeted digit. Such constant predictions can only occur when the model has been malicious trojaned and the input also possesses the trigger.

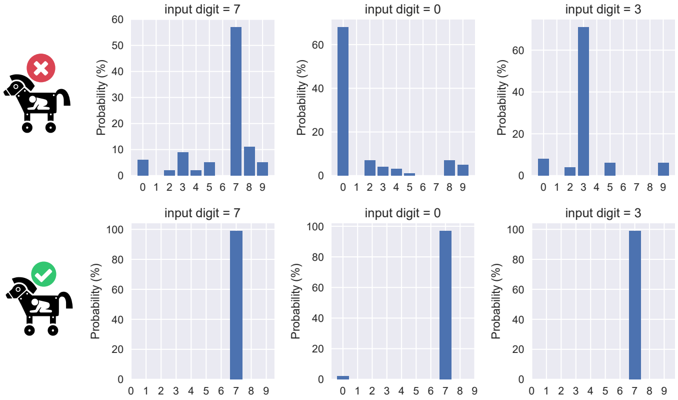

Figure 5. Predicted digits' distribution of 1000 perturbed images applied to *one given input image*. The y-axis is "class label". The inputs of top three sub-figures are trojan-free. The inputs of bottom sub-figures are trojaned. The attacker targeted class is 7.

perturbation considered in this work is image linear blend—superimposing two images. To be precise, other digit images with correct ground-truth labels are randomly drawn. Each of the drawn digit image is then linearly blended (superimposed) with the incoming input image. Noting other perturbation strategies besides the specific image superimposing mainly utilized in this work can also be taken into consideration. Under expectation, the predicted numbers (labels) of perturbed inputs vary significantly when linear blend is applied to the incoming clean input image. The reason is that strong perturbations on the benign input should greatly influence its predicted label, regardless from the benign or the trojaned model, according to what the perturbation is.

In Fig. 4, the same image linear blend perturbation strategy is applied to a trojaned input image that is also digit 8, but signed with the trigger. In this context, according to the aim of the trojan attack, the predicted label will be dominated by the trojan trigger—output class becomes input-agnostic. Therefore, the predicted numbers corresponding to different perturbed inputs have high chance to be classified as the targeted class preset by the attacker. In this specific exemplified case, the predicted numbers are always 7. This abnormal behavior violates the fact that the model prediction should be input-dependent for a benign model. Thus, we can come to the conclusion that this incoming input is trojaned.

Fig. 5 depicts the predicted classes' distribution given that 1000 random drawn digit images are linearly blended with one given incoming benign or trojaned input. Top sub-figures are for benign digit inputs (7, 0, 3 from left to right). Digit inputs at the bottom are still 7, 0, 3 but trojaned. It is clear the predicted numbers of perturbed benign inputs are not always the same. However, the predicted numbers of perturbed trojaned inputs are always the constant. Overall, high randomness of the predicted numbers of perturbed inputs potentially indicates a benign input; whereas low randomness means a trojaned input.

IV. STRIP Trojan Detection System Design

We now firstly lay out an overview of our STRIP trojan detection system that is augmented with the returned (trojaned) model under deployment. Then we define the adversarial model considered by us, followed by two metrics of evaluating detection performance. We further formulate the way of assessing the randomness using entropy for a given incoming input. This helps to facilitate the determination of a trojaned/clean input.

A. Detection System Overview

The run-time STRIP trojan detection system is depicted in Fig. 6 and further illustrated in Algorithm 1. The perturbation step generates N perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ corresponding to **one** given incoming input x. Each perturbed input is a superimposed image of the input x (replica) and an image randomly drawn from the user held-out testing dataset, $\mathcal{D}_{test}$. All the perturbed inputs along with x itself are concurrently fed into the deployed DNN model, $F_\Theta(x_i)$. According to the input x, the DNN model predicts its label z. At the same time, the DNN model determines whether the input x is trojaned or not based on the observation on predicted classes to all N perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ that forms a set $\mathcal{D}_p$. In particular, the randomness (entropy), as will be detailed soon in Section IV-D, of the predicted classes is used to greatly facilitate the judgment on whether the input is trojaned or not.

Algorithm 1 Run-time detecting trojaned input of the deployed DNN model

```
procedure detection ($x$, $\mathcal{D}_{test}$, $F_\Theta()$, detection boundary )
    trojanedFlag ← No
    for $n = 1 : N$ do
        randomly drawing the $n_{\text{th}}$ image, $x_n^t$, from $\mathcal{D}_{test}$
        produce the $n_{\text{th}}$ perturbed images $x^{pn}$ by superimposing incoming image $x$ with $x_n^t$.
    end for
    $\mathbb{H} \leftarrow F_\Theta(\mathcal{D}_p)$    ▷ $\mathcal{D}_p$ is the set of perturbed images consisting of $\{x^{p_1}, ......, x^{p_N}\}$, $\mathbb{H}$ is the entropy of incoming input $x$ assessed by Eq 4.
    if $\mathbb{H} \leq$ detection boundary then
        trojanedFlag ← Yes
    end if
    return trojanedFlag
end procedure
```

This run-time STRIP trojan detection is agnostic to a specific DNN model/architecture. Yet simple, it is surprisingly straightforward and easy to be implemented, requiring neither costly computational resources nor time-consuming examinations of the model parameters—from a white-box access perspective.

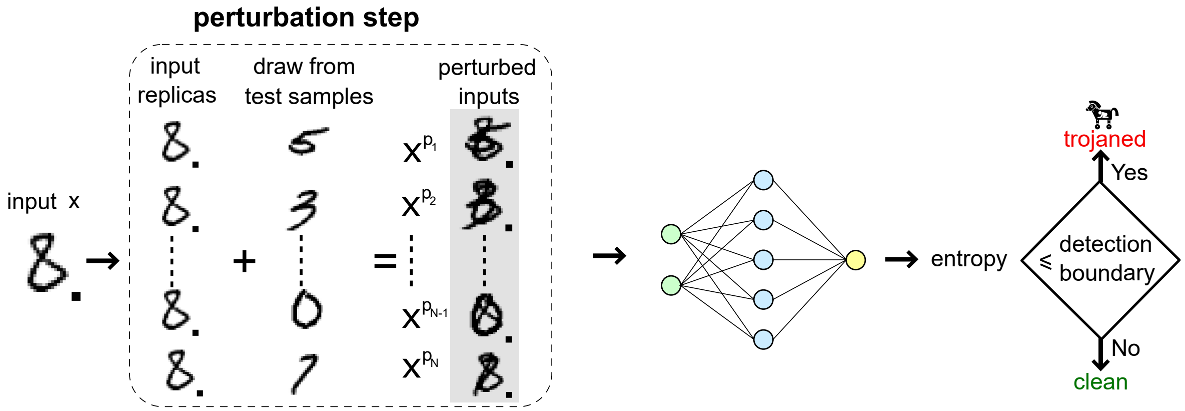

Figure 6. Run-time STRIP trojan detection system overview. The input x is replicated N times. Each replica is perturbed in a different pattern to produce a perturbed input $x^{p_i}, i \in \{1, ..., N\}$. According to the randomness (entropy) of the predicted labels of the perturbed replicas, whether the input x is a trojaned input is easily determined.

B. Adversarial Model

The attacker's goal is to return a trojaned model with its accuracy performance comparable to that of the benign model for clean inputs. However, its prediction is hijacked by the attacker when the trigger that is only known to the attacker is presented. As a defense work, we consider that an attacker has maximum capability. We assume that the attacker has full access to the training dataset and white-box access to the DNN model/architecture. The attacker determines and inserts whatever trigger with respect to e.g., location, size, he/she prefers. The attacker returns the trained DNN model to the user for deployment.

For the defender side, same to [12], [14]. we reason that he/she has held out a small collection of benign samples with correct labels. However, the defender does not have access to trojaned data stamped with triggers. We argue that the attacker is extremely unlikely to ship the poisoned training data to the user. Not surprisingly, this reasonable assumption eventually implies that one concurrent countermeasure [15] is less practical as it does require access to the poisoned training data by the defender.

C. Detection Capability Metrics

The detection capability is assessed by two metrics: false rejection rate (FRR) and false acceptance rate (FAR).

1) The FRR is the probability when the benign input is regarded as a trojaned input by the STRIP detection system.
2) The FAR is the probability that the trojaned input is recognized as the benign input by the STRIP detection system.

In practice, the FRR stands for robustness of the detection, while the FAR usually introduces the security concerns. Ideally, both FRR and FAR should be 0%. This condition may not always possible in reality. Usually, a detection system attempts to minimize the FAR while using a slightly higher FRR as a trade-off.

D. Entropy

We consider Shannon entropy to express the randomness of the predicted labels of all perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ corresponding to a given incoming input x. Starting from the n_{th} perturbed input $x^{p_n} \in \{x^{p_1}, ..., x^{p_N}\}$, its entropy $\mathbb{H}_n$ can be expressed:

$$\mathbb{H}_n = -\sum_{i=1}^{i=M} y_i \times \log_2 y_i \tag{2}$$

with y_i the probability of the perturbed input belonging to class i. M is the total number of classes, defined in Section II-A.

Based on the entropy $\mathbb{H}_n$ of each perturbed input x^{p_n}, the entropy summation of all N perturbed inputs $\{x^{p_1}, ..., x^{p_N}\}$ can be expressed:

$$\mathbb{H}_{\text{sum}} = \sum_{n=1}^{n=N} \mathbb{H}_n \tag{3}$$

with $\mathbb{H}_{\text{sum}}$ stands for the chance the input x being trojaned. Higher the $\mathbb{H}_{\text{sum}}$, lower the probability the input x being a trojaned input.

We further normalize the entropy $\mathbb{H}_{\text{sum}}$; the normalized entropy is written as:

$$\mathbb{H} = \frac{1}{N} \times \mathbb{H}_{\text{sum}} \tag{4}$$

To this end, $\mathbb{H}$ is regarded as the entropy of one incoming input x. It serves an indicator whether the incoming input x is trojaned or not.

V. Evaluations

A. Experiment Setup

We evaluate on two vision applications: hand-written digit recognition based on MNIST [16] and image classification based on CIFAR10 [17]. Both of them are using convolution neural network, which is popular in computer vision applications. Dataset and model architecture details are summarized in Table I. We avoid complicated model architectures to relax the computational overhead, thus, expedite evaluations. The

Table I
DETAILS OF MODEL ARCHITECTURE AND DATASET.

Dataset	# of labels	image size	# of images	Model Architecture	Total Parameters
MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense	80,758
CIFAR10	10	$32 \times 32 \times 3$	60,000	8 Conv + 3 Pool + 3 Dropout 1 Flatten + 1 Dense	308,394

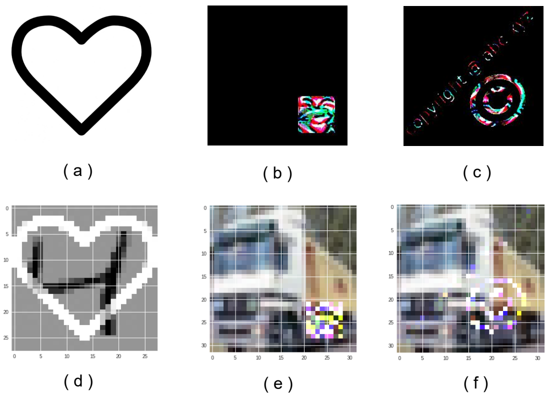

Figure 7. Besides the square trigger shown in Fig. 2. Other three trojan triggers and their corresponding trojaned input samples used in case studies. Triggers in (b) and (c) were used in NDSS 2018 [20] to perform trojan attacks and IEEE S&P 2019 [14] to demonstrate defense. The heart shape graffiti created by us is used to demonstrate STRIP detection's insensitivity to trigger sizes, which overcomes the limitation of IEEE S&P 2019 [14] and Standford work [12].

STRIP trojan detection is not only suitable for vision domain that is the focus current work but also applicable to text and speech domains [18], [19]. In those domains, instead of image linear blend used in this work, other specific perturbations can be considered. For example, in the text domain, one can randomly chunk out a small fraction of text to observe the predictions. If the text is trojaned, predictions should be constant, because most of the time the trojan trigger will not be removed. In text domain, it is worth to mention that the space for inserting the trigger is limited in comparison with the image domain since the attacker needs to reserve the semantics of the text.

Besides the square trigger shown in Fig. 2, we also use other three triggers shown in Fig. 7. Notably, the triggers used in this paper are those that have been used to perform trojan attacks in [8], [20] and also used to evaluate countermeasures against the trojan attack in [12], [14]. Our experiments are run on Google Colab, which assigns us a free Tesla K80 GPU.

B. Case Study: MNIST

For MNIST dataset, the square trigger shown in Fig. 2 and the heart trigger in Fig. 7 (a) are used. The square trigger only occupies nine pixels—trigger size is 1.15% of the image, while the heart shape is resized to be the same size, 28×28, of the digit image.

We have tested 2000 clean digits and 2000 trojaned digits. Given each incoming digit x, $N = 100$ different digits randomly drawn from the held-out test samples are linearly blended with x to generate 100 perturbed images. Then the entropy of the input x is calculated according to Eq 4 after feeding all 100 perturbed images to the deployed model. The entropy distribution of the tested 2000 benign and 2000 trojaned digits are depicted in Fig. 8 (a) (with the square trigger) and Fig. 8 (b) (with the heart trigger).

We can observe that the entropy of a clean input is always large, while the entropy of the trojaned digit is always small. Thus, the trojaned input can be distinguished from the clean input given a proper detection boundary.

C. Case Study: CIFAR10

Table II
ATTACK SUCCESS RATE AND CLASSIFICATION ACCURACY OF TROJAN ATTACKS ON TESTED TASKS.

Dataset	Trigger type	Trojaned model		Origin clean model classification rate
		Attack success rate	Classification rate	
MNIST	square (Fig. 2)	98.86%	99.86%	98.62%
MNIST	trigger a (Fig. 7 (a))	98.86%	100%	98.62%
CIFAR10	trigger b (Fig. 7 (b))	87.23%	100%	88.27%
CIFAR10	trigger c (Fig. 7 (c))	87.34%	100%	88.27%

As for CIFAR10 dataset, the triggers shown in Fig. 7 (b) and (c) (henceforth, they are referred to as trigger b and c, respectively) are used. Again, like the two triggers used for MNIST, one trigger size is small, while the other trigger size is large. The purpose is to demonstrate the efficacy of the STRIP detection method, which is independent to the trigger size, as long as the trigger has a salient effect to hijack the prediction.

We also tested 2000 benign and trojaned input images, respectively. Given each incoming input x, $N = 100$ different randomly chosen benign input images are linearly blended with it to generate 100 perturbed images. Then the entropy corresponding to each input image is calculated according to Eq 4. The entropy distribution of the tested 2000 benign and 2000 trojaned input images are depicted in Fig. 8 (c) (with trigger b) and Fig. 8 (d) (with trigger c), respectively. Under expectation, the entropy of benign input image is always large, while the entropy of the trojaned input image is always small. Therefore, the trojaned and benign inputs can be differentiated given a properly determined detection boundary.

D. Detection Capability: FAR and FRR

To evaluate the FAR and FRR, herein we assume that we have access to trojaned inputs in order to estimate their corresponding entropy values (we can pretend to be an attacker in our evaluation). However, in practice, *the defender is not supposed to have access to any trojaned samples* under assumptions of our adversarial model, see Section IV-B. So one may ask:

How the user is going to determine the detection boundary?

Given that the model has been returned to the user, the user has arbitrary control and access to the model and the held-out test samples with correct ground-truth label. The user can estimate the entropy distribution of benign inputs. It is reasonable to assume such distribution is a normal distribution,

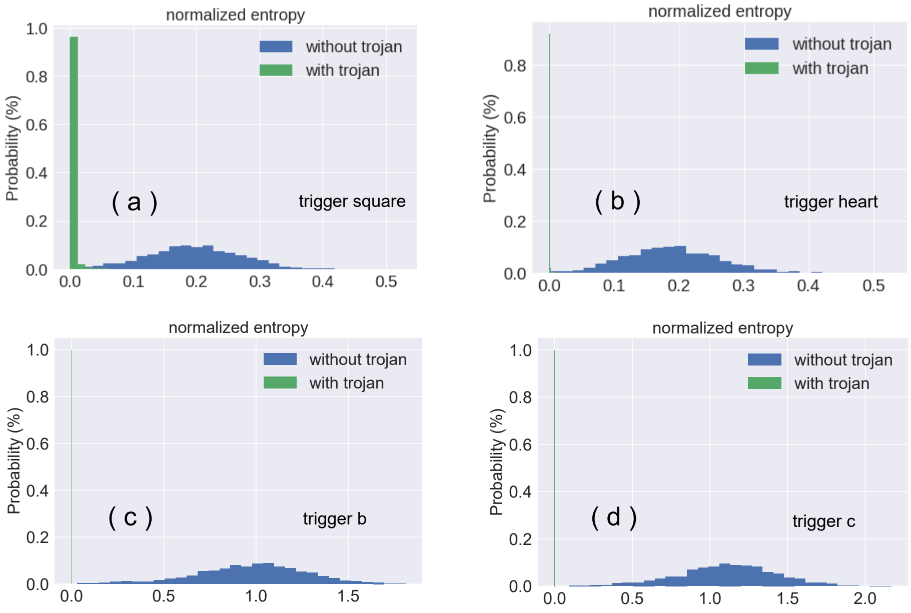

Figure 8. Entropy distribution of benign and trojaned inputs. The trojaned input always has a small entropy, which can be winnowed given a proper detection boundary (threshold). (a) The trigger is the square. Dataset is MNIST. (a) The trigger is the heart shape graffiti. Dataset is MNIST. (a) The trigger is the trigger b. Dataset is CIFAR10. (a) The trigger is the trigger c. Dataset is CIFAR10.

Table III
FAR AND FRR OF THE STRIP TROJAN DETECTION SYSTEM.

Dataset	trigger type	N	mean	standard variation	FAR	detection boundary	FRR
MNIST	square, Fig. 2	100	0.196	0.074	3% 2% 1%[1]	0.058 0.046 0.026	0.75% 1.1% 1.85%
MNIST	trigger a, Fig. 7 (a)	100	0.189	0.071	2% 1% 0.5%	0.055 0.0235 0.0057	0% 0% 1.5%
CIFAR10	trigger b, Fig. 7 (b)	100	0.97	0.30	2% 1% 0.5%	0.36 0.28 0.20	0% 0% 0%
CIFAR10	trigger c, Fig. 7 (c)	100	1.11	0.31	2% 1% 0.5%	0.46 0.38 0.30	0% 0% 0%

[1] When the FAR is set to be 0.05%, the detection boundary value becomes a negative value. Therefore, the FRR given FAR of 0.5% does not make sense anymore, which is not evaluated.

which has been affirmed in Fig. 8. Then, the user gains the mean and standard variation of the normal entropy distribution of benign inputs. The FRR, e.g., 1%, of a detection system can always be firstly determined. Then the percentile of the normal distribution can be acquired. *This percentile is actually the detection boundary chosen by the user*. In other words, for the normal distribution of the benign input entropy, this detection boundary (percentile) results in a FRR of 1%. The FAR is the probability that the entropy of an incoming trojaned input is larger than this detection boundary. In this context, we evaluate the fraction of trojaned inputs that their entropy are larger than the chosen detection boundary out of all tested trojaned inputs.

Notably, in contrast to the use of a global detection boundary [14], the detection boundary in our detection system is specific to each deployed model, which is not a global setting. This avoids the potential cumbersome that a global setting may fail to a specific model as the optimized detection boundary to each model will vary.

Table III summarises the detection capability for four different triggers on both the MNIST and CIFAR10. It is not surprising that there is a tradeoff between the FAR and the FRR—the FAR increases with the increase of FRR. In our case studies, the chosen of a 1% FRR suppresses the FAR to be always less than 1%, which is acceptable in practice. If the security concern is extremely high, the user can chose a larger FRR to obtain a detection boundary that further suppresses the FAR.

For the CIFAR10 dataset with the trigger either b or c, we actually always get 0% FAR. Therefore, we take a look at the minimum entropy of the tested 2000 benign inputs and the maximum entropy of the tested 2000 trojan inputs. We find that the former is greatly larger than the latter. To be precise, a 0.029 for the minimum clean input entropy and 7.74×10^{-9} for the maximum trojan input entropy when the trigger b is used. While a 0.092 for the minimum clean input entropy and 0.005 for the maximum trojan input entropy when the trigger c is used. We can see that there exists a large entropy gap between the benign input and trojaned input, which explains the 0% FAR.

Next, we investigate the detection capability as a relationship with the depth of the neural network—relevant to the accuracy performance of the DNN model. We also investigate the time overhead for executing the detection.

1) Depth of Neural Network: Besides the DNN architecture—referred to as 8-layer architecture—achieving around 88% accuracy performance for clean inputs, we tested a shallow neural network architecture only with 2 conventional layer and 1 dense layer—referred to as 2-layer architecture. For this 2-layer architecture, the benign model on CIFAR10 dataset has a lower accuracy performance, which is 70%. The corresponding trojaned model with trigger c has a similar accuracy with around 70% for clean inputs while around 99% attack success rate for trojaned inputs. In this context, it is still reasonable to say that the trojaned behavior is successfully inserted into the model based on the 2-layer architecture such that the trojaned model does not degrade the performance on clean inputs.

We find that as the neural network goes deeper—usually leads to a more accurate prediction, the detection capability also improves. Specifically, for the shallow 2-layer architecture based trojaned model, a 2% FRR gives a 0.45% FAR, a 1% FRR gives a 0.6% FAR, and a 0.5% FRR gives a 0.9% FAR. While for the 8-layer architecture based trojaned model, the FRR is always 0%, regardless of the FRR. This is because there is always an entropy gap—no overlap—between the benign and trojaned inputs.

Moreover, we run a 8-layer architecture on the MNIST dataset with the square trigger. For the trojaned model, its accuracy on clean input is 99.02% while achieves a 99.99% accuracy on trojaned input. Our STRIP demonstrates an improved detection capability as well. Specifically, a 1% FRR gives a 0% FAR, a 0.5% FRR gives a 0.03% FAR, which has been greatly improved in comparison with the STRIP detection capability to a 2-layer trojaned model, see Table. III.

To this end, we can empirically conclude that the deeper the model, the higher detection capability of the STRIP detection. On one hand, this potentially lies on the fact the model with more parameters memorizes the trigger feature even stronger, which always present a low entropy for the trojaned input. On the other hand, the model also more accurately memorizes the features for each class of clean input. The trained model is more sensitive to strong perturbation on clean input, and therefore, unlikely to present a low entropy—may contribute to FRR.

We are curious on those images that are trojaned but falsely accepted as clean images. Therefore, based on the 2-layer trojaned model (8-layer model has 0% FAR) produced on the CIFAR10 dataset and trigger c, we further examine those images. We found that most of them have already lost their trojan behavior, as shown in Fig. 9. For instance, out of 10 falsely accepted trojaned images, four images maintaining their trojaning effect of hijacking the DNN model to classfy them to be the targeted label of ‘horse’. The rest six trojaned images are unable to achieve their trojaning effect because the trojan trigger is not strong enough to misdirect the predicted label to be ‘horse’. In other words, these six trojaned images will not cause security concerns desinged by the attacker when they are indeed misclassified into benign image by STRIP. In addition, we observe that there are three trojaned images classified into their correct ground-truth labels by the attacker’s trojaned model. The reason may lie on that the trigger feature is greatly weakened in certain specific inputs. For example, without careful attention, one may not perceive the stamped trigger in the ‘frog’ (1st) and ‘airplane’ (7th) images, which is more likely the same to the trojaned DNN model.

2) Detection Time Overhead: Fig. 11 depicts relationship between the detection time latency and the N—number of perturbed inputs. In practice, it is unnecessary to always adopt a larger N on the condition that the FAR has already been significantly minimized. In this regard, we vary the N from 2 to 100 to observe the detection capability. Actually, when N is around 10, the maximum trojan input entropy is always less than the minimum benign input entropy (CIFAR10 dataset with trigger c). This already ensures both FRR and FAR to be zero if the user picks up the minimum benign input entropy as the detection boundary.

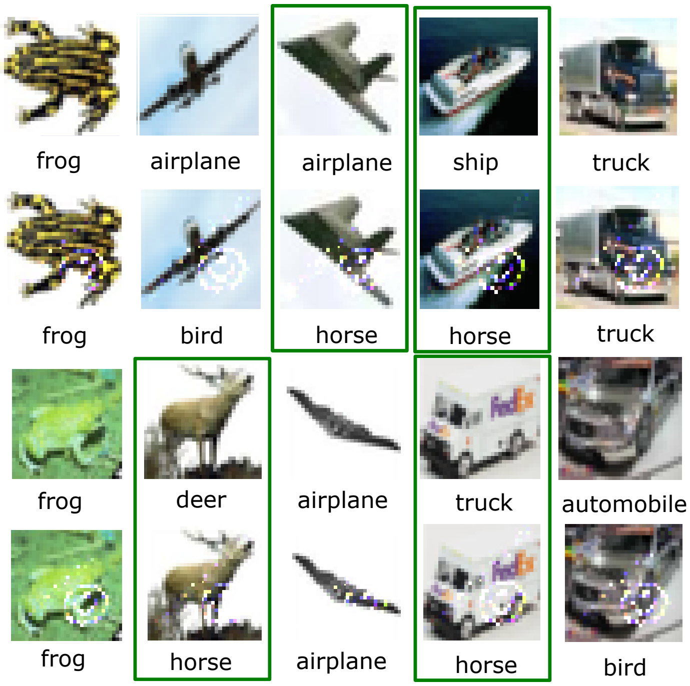

Figure 9. When the trojaned images are falsely accepted by the STRIP as benign images, most of them actually have lost their trojaning effect. Because they cannot hijack the trojaned DNN model to classify them to the targeted class—‘horse’. Green-boxed trojaned images are those ultimately bypassing STRIP detection system while maintaining there trojaning effect.

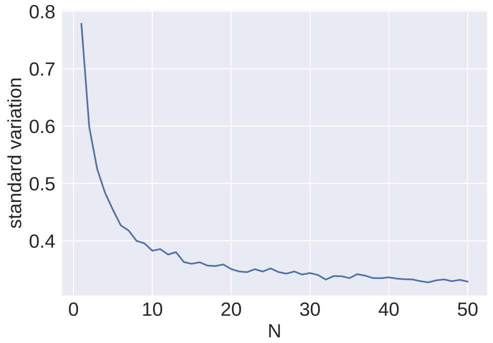

Figure 10. The relationship between the standard variation of the benign input entropy distribution and the N, with N the number of perturbed patterns. In practice, the defender can rely on this to properly select N. Choosing the N when the slope of standard variation starts not decreasing too much. Our empirical results suggests N beyond 8 is adequate. Further increasing N does not increase the detection capability too much. The trigger is the trigger c. Dataset is CIFAR10.

Properly selecting a smaller N reduces the time latency for detecting the trojaned input during run-time. This is imperative for real-time applications such as traffic sign recognition. To this end, one may rise the following question:

How to determine the N by only relying on the normal distribution of the benign input entropy?

Our solution is to observe the change of the standard variation of the benign input entropy distribution as a function of N. One example is shown in Fig. 10. The user can gradually increase N, when the slope of standard variation starts not decreasing too much, the user can pick up this N.

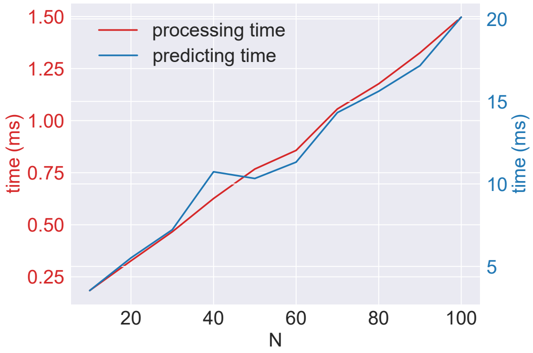

Figure 11. The relationship between the detection time overhead and the N. Detection time including the processing time and predicting time. Processing time is the time used to prepare N perturbed images given an incoming input x, while predicting time is the time used to predict the label of those N perturbed images and obtain the entropy of input x. The trigger is the trigger c. Dataset is CIFAR10.

According to our empirical case studies on CIFAR10 dataset, setting $N = 10$ is sufficient, this also confirms the above N selection methodology as shown in Fig. 10. Without optimization, it is 1.5 times higher than the time of the original inference time . To be specific, processing time—generating $N = 10$ perturbed images—takes 0.18ms, predicting one image costs 2.38ms, while predicting 10 images takes 3.48ms. In total, the STRIP detection overhead is 3.66ms, while the original inference time without implementing the detection is 2.38ms. If the real-time performance requiring achieving the original inference time when plugging the STRIP detection system is highly critical, parallel computation can be taken into consideration. Given parallel computation power is sufficient; the detection time can be ultimately reduced to comparable to the original prediction time. This is because image processing—it has been done before feeding perturbed images into the DNN model—time is very small. Noting the 0.18ms processing time is when we sequentially generate those 10 perturbed images, this generation can be paralleled. In addition, the prediction of N perturbed images can be run independently and in parallel.

VI. Discussion

Multiple Models from Different Providers. Under the assumption that different model providers are not collusive, we suggest outsourcing the training procedure of a DNN model to different providers, and then deploying the returned models in parallel to the security-critical applications (e.g., face recognition to gain access control). This assumption is reasonable, especially for competitive providers. Herein, we do not assume that the providers are honest. Therefore, they may insert a backdoor into the returned models. But they are likely to set different triggers because the triggers are arbitrary and are only known by the attackers.

In this context, a user can choose to deploy multiple models returned from different model providers and plug STRIP detection system to each model. For instance, consider three outsourced models are returned as: A, B and C. In the worst case, three models are all trojaned but with different triggers: a, b and c. Given a benign input, these three detection systems tend to give the same correct label and none throws a ‘trojan’ alert. As a consequence, the whole system deems this input as benign. When the input is with a trigger (e.g., trigger a), model A will misclassify the input into an incorrect class. However, model B and model C still regard this input as benign and correctly predict the label—models B and C are not infected by trigger a. Based on the majority voting principle, the whole system can proceed by using the agreed label from detection system B and C. At the same time, we can receive a trojan alert from our defense on model A, which provides a twofold meaning: the current input is trojaned, and model A has a high chance being trojaned.

The benefits of setting of deploying multiple models from different providers are:

1) Correct decision can still be made even all DNN models are trojaned. This facilities ML in those saft-critical applications that requires rare or no human interference
2) The malicious input along the corresponding DNN model are marked during run-time but without stopping the application, which will be examined off-line to identify and further removal the trigger behaviors.

Trojan Detection and Mitigation. From the related work—detailed in Section VII, it is observed that initial studies of countering trojan attacks execute trojan mitigation/removal based on an assumption that all models under deployment are trojaned. However, this tends to be problematic in practice since, in most situations, DNN models returned from model providers are benign—most providers are trustworthy. It is unreasonable to perform trojan behavior removal operations on benign models. Trojan mitigation requires extensive computation and time overhead. Moreover, blindly executing mitigation operations without the knowledge of the trojan trigger can result in deterioration of the prediction accuracy for benign inputs.

Therefore, trojan behavior detection shall be considered as a prerequisite before conducting trojan behavior removal. The main benefit is that the trojan mitigation can be executed more confidently to avoid the time and computational costs on those benign models. Our detection system tells whether an input/model is trojaned or not. If the input is judged to be a trojaned input, the user can further confirm its targeted label (e.g., most likely, the constantly predicted classes of those perturbed images). We can further utilize the approach presented in [14] to reverse engineer the trigger. Knowing the targeted label will dramatically decrease the efforts to identify the trigger [14], whereas in [14] one potential trigger is firstly reversed corresponding to each class and the highest ranked trigger is treated as the potential trigger used by the attacker. Built upon the reversed trigger, there are several available techniques to mitigate or remove the trojan behavior, such as i) filtering the trojaned input, ii) patching the DNN model via pruning and iii) patching the DNN model via unlearning that can remove the trojan behavior from the trojaned DNN model while preserving the model performance [14], [21].

Explainable Decisions. The STRIP detection system presented in this paper is not the end of trojan attacks in the ever-competitive security race. Lack of interpretability is eventually what an trojan attacker exploits. To trust the decisions made by DNN models in security-critical scenarios, it is imperative for the DNN models to provide explanations which are meaningful for humans to understand the model outputs, as initially highlighted and investigated in CCS 2018 [5]. For examples, given an input, the model shall identify which features contribute the most to the model outputs [5], [22], [23]. More aggressively, one expects to identify what features cause the model to predict a specific label. Addressing the above two explanations, especially the later, can eventually help us to substantially alleviate the trojan attacks.

Given an inferred class, if a model can explain which are the key features leading to it, the user can analyse the dominant features for each label and discover the abnormal triggering feature if the model is trojaned. Second, if a model can identify the most impacting features with respect to the model outputs, the explanations can be integrated into our defence to strengthen it. Specifically, once we judge the input is trojaned via the STRIP detection system, we can further check what is the most dominant feature resulting in misclassification for the system. This feature shall be governed by a trigger. Therefore, the feature of the trigger itself can be discovered. We can use this feature as the trigger—the extracted feature may not exactly same to the trigger itself but illustrates its salient e.g., pixels or shape [5], then pruning and patching on the trojaned model can be applied to mitigate the inserted trojan behavior [14], [21]. We leave detailed investigations to future work.

VII. Related Work

Trojan attack is an emerging attack on DNN models, which is one insidious variant of poisoning attacks considering the requirement of manipulating training data. It allows the attacker to easily craft adversarial examples in the physical world to perform targeted attacks. Previous work on poisoning attacks usually aim to degrade a classifier's accuracy to clean inputs [24], [25]. In contrast, trojan attack maintains a prediction accuracy for clean inputs as high as a benign model, while misdirecting the input to a targeted class whenever the input is trojaned with an attacker-chosen trigger.

A. Attack

In 2017, Gu *et al.* [8] propose Badnets, where the attacker has access to the training data and thus can manipulate the training data to insert an arbitrarily chosen trigger and also change the class labels. Gu *et al.* [8] use a square-like trigger located in the right corner of the digit image of the MNIST data—we also use this trigger type—to demonstrate the trojan attack. On MNIST dataset, the authors demonstrate an attack success rate of over 99% without impacting model performance on benign inputs. Our attack has a similar performance. In addition, trojan triggers to misdirect traffic sign classifications have also been investigated in [8], [11]. Chen *et al.* [6] from UC Berkeley concurrently demonstrate such backdoor attack by poisoning the training dataset.

Liu *et al.* [20] eschew the requirements of accessing the training data. Instead, their attack is performed during model update phase, not model training phase. They first carry out a reverse engineer to gain the training data, then improve the trigger generation process by delicately designing triggers to maximum the activation of chosen internal neurons in the neural network. This builds a stronger connection between triggers and internal neurons, thus, requiring less training samples to insert effective trojans.

Bagdasaryan *et al.* [10] show that federated learning is fundamentally vulnerable to trojan attacks. Firstly, participants are enormous, e.g., millions, it is impossible to guarantee that none of them are malicious. Secondly, federated learning is designed to have no access to the participant's local data and training process to ensure the privacy of the sensitive training data; therefore, participants can use trojaned data for training. The authors demonstrate that with controlling no more than 1% participants, an attacker is able to cause a global model to be trojaned and achieves a 100% accuracy on the trojaned input even when the attacker is only selected in a single round of training—federated learning requires a number of rounds to update the global model parameters. This federated learning trojan attack is validated through CIFAR-10 dataset. We chose the same dataset to demonstrate our countermeasure in this paper.

B. Defense

Though there are general defenses against poisoning attacks [26], they cannot be immediately mounted to against trojan attacks. In practice, the user has no knowledge of the trojan trigger and no access to trojaned training sample, which make combating trojan attacks more challenging.

The countermeasures in [21], [27] suggest approaches to remove the trojan behavior without frist check whether the model has been potentially trojaned. Fine-tuning is used to remove potential trojans by pruning dedicatedly chosen parameters of the DNN model [21]. However, this method substantially degrades the model accuracy [14]. It is also cumbersome to perform removal operation to any DNN model as most of them maybe benign. Approaches presented in [27] incur high complexity and computation costs.

Chen *et al.* [15] propose an activation clustering (AC) method to detect whether the training data has been trojaned or not prior to the deployment. The intuition behind this method is that the reasons why the trojaned and the benign samples receive the same predicted label by the trojaned DNN

Table IV
COMPARISON WITH OTHER TROJAN DETECTION WORKS.

Work*	Black/White -Box Access[1]	Run-time	Computation cost	Time overhead	Trigger size dependent	Access to Trojaned data	Detection capability
Activation Clustering (AC), Chen *et al.* [15]	White-box	No	Moderate	Moderate	No	Yes	F1 score nearly 100%[2]
Neural Cleanse, Wang *et al.* [14]	Black-box	No	High	High	Yes	No	100%[3]
SentiNet, Chou *et al.* [12]	Black-box	Yes	Moderate	Moderate	Yes	No	5.74% FAR and 6.04% FRR[4]
STRIP, Ours	Black-box	Yes	Low	Low	No	No	0.46% FAR and 1% FRR[4]

* AC and Neural Cleanse are performed offline prior to the model deployment to detect whether the model has been trojaned or not. The SentiNet and STRIP are performed during run-time checking incoming input to see whether the input is trojaned or not.

[1] White-box requires access to inner neurons of the model.

[2] It detects whether the training data has been trojaned or not by assuming that the attacker would send the poisoned training data back to the user.

[3] According to case studies on 6 infected, and their matching original model, the authors [14] show all the infected/trojaned and clean model can be clearly distinguished.

[4] The average FAR and FRR or SentiNet and STRIP are on different datasets. We will run SentiNet method on MNIST and CIFAR10 for a fair comparison with it after its code release [12]: the authors will release their code after publication.

model are different. By observing neuron activation of benign samples and trojaned samples that produce the same label in hidden layers, one can potentially distinguish trojaned samples from legitimate samples via the activation difference. However, this method requires that the user has access to the trojaned training data in hand, which appears to be unrealistic. It is very unlikely that the attacker will ship his/her trojaned data samples to the user given that the attacker has performed the trojaned attacks.

Chou *et al.* [12] from Standford exploit both the model interpretability and object detection techniques, referred to as SentiNet, to firstly discover a contiguous regions of an input image. This region is assumed as having a high chance of possessing a trojan trigger when it strongly affects classification. Once this region is determined, it is carved out and patched to other held-out images that are with ground-truth labels. If the misclassification rate—probability of the predicted class is not the ground-truth label of the held-out image—of these patched images are high enough, then this carved patch is regarded as an adversarial patch that eventually contains a trojan trigger. Therefore, the incoming input is an adversarial input. We regard this concurrent work is mostly related to ours since both SentiNet and our STRIP focus on detecting whether the incoming input has been trojaned or not during run-time. However, there are at least two main differences: i) we do not care about the ground-truth label of neither the incoming input nor the drawn images from the held-out samples, while [12] relies on the ground-truth label of the held-out images; ii) we constructively introduce the entropy to evaluate the randomness of the incoming input, which is more convenient, straightforward and easy-to-use in comparison with the evaluation methodology presented in [12]. Most importantly, as the author stated, one inherent limitation of [12] is that the region embeds the trojan trigger must be small enough. If the trigger region is largely alike, such as the trigger shown in Fig. 7 (a) and (c), or those in [11] used to perform attacks that are sticks spread over the image, then SentiNet becomes ineffective. This is caused by its carve-out method. Supposing that the carved region is large and contains the trojaned trigger, then patching it on held-out samples will also show a small misclassification rate to be falsely accepted as a benign input via SentiNet. Our STRIP naturally overcomes this issue.

Wang *et al.* [14] in 2019 S&P propose a Neural Cleanse method to detect whether a DNN model has been trojaned or not prior to deployment. Neural Cleanse is based on the intuition that, given a backdoored model, it requires much smaller modifications to all samples to make them being misclassified into the attacker targeted (infected) label than any other uninfected labels. Therefore, their method iterates through all labels of the model and determine if any label requires a substantially smaller amount of modification to achieve misclassification. One advantage of this method is that the trigger can be reversed and thus discovered during the trojaned model detection process. However, this method has two noticeable limitations. Firstly, it could incur high computation costs proportional to the number of labels. The computation cost of detection process can take up to several days for certain DNN models even when the optimization is adopted. Secondly, similar to SentiNet [12], it becomes ineffective for large size triggers.

Comparison: We compare with other three trojan detection works as detailed in Table IV. Noting that AC and Neural Cleanse are performed offline prior to the model deployment to *directly detect whether the model has been trojaned or not*. The SentiNet and STRIP are performed during run-time checking incoming input to *detect whether the input is trojaned or not when the model is already deployed*. Our method is efficient in terms of computation cost and time overhead. While AC and our STRIP are insensitive to trojan trigger size, the AC appears to be impractical in reality as it assumes the trojaned data is in hand.

C. Watermarking

There are works considering a backdoor as a watermark [28] to protect the intellectual property (IP) of the trained DNN model [29]–[31]. The argument is that the inserted backdoor can be used to claim the ownership of the model provider since only the provider is supposed to have the knowledge of such backdoor, while the backdoored DNN model has no (or imperceptible) degraded functional performance on normal inputs. However, as the above countermeasures—detection, recovery, and removal—against backdoor insertion are continuously evolved, the robustness of using backdoors as watermarks is potentially challenged in practical usage. It is important to note that defeating watermarking is out of the scope of our work. We leave the robustness of backdoor entangled watermarking under the backdoor detection and removal threat as part of the future work.

VIII. Conclusion

The presented STRIP advances the front-line defense against recently revealed trojan attacks on DNN models, which constructively turns the strength of such attacks into a weakness. The input-agnostic characteristic allows us to detect the trojaned input on run-time and treats the returned model as a black-box without examining mega parameters of the DNN model.

Ultimately, we enforce the attacker under a dilemma situation; the stronger the trojan trigger behaves in the physical world, the easier it is to be detected. Experiments on MNIST and CIFAR10 datasets with trojan triggers identified in previous trojan attack works affirm the high detecting capability of the presented perturbation methodology. Overall, the FAR is suppressed to be lower than 1% in given a user tolerable FRR of 1%. The 0% FRR and 0% FAR are empirically achieved on the popular CIFAR10 dataset. While easy-to-implement, time-efficient and complementing to existing trojan mitigation techniques, the run-time STRIP works in a black-box manner and overcomes one limitation of other state-of-the-art detection methods that is sensitive to trigger size (ineffective to detect trojan trigger of large size).

Although we have demonstrated high efficacy of STRIP against trojan attacks on DNN based computer vision applications. As future work, it is interesting to test its generalization to e.g., text and voice domain (we think the STRIP is also applicable), investigate how STRIP is efficient and what other specific strong intentional perturbation strategies can be adopted. As the security-race is continuously evolving, it is also necessary to evaluate STRIP on potential upcoming variants of trojan attacks.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia II, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1145–1153.

[3] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016, pp. 258–263.

[4] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for AI," *arXiv preprint arXiv:1712.05855*, 2017.

[5] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 364–379.

[6] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[7] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 349–363.

[8] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[9] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "Potrojan: powerful neural-level trojan designs in deep learning models," *arXiv preprint arXiv:1802.03043*, 2018.

[10] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[11] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[12] E. Chou, F. Tramèr, G. Pellegrino, and D. Boneh, "Sentinet: Detecting physical attacks against deep learning systems," *arXiv preprint arXiv:1812.00292*, 2018.

[13] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1528–1540.

[14] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proceedings of the 40th IEEE Symposium on Security and Privacy*, 2019.

[15] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[18] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[19] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[20] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Network and Distributed System Security Symposium (NDSS)*, 2018.

[21] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proceedings of RAID*, 2018.

[22] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1135–1144.

[23] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017, pp. 4765–4774.

[24] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[25] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[26] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: A data provenance based approach," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 103–110.

[27] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 45–48.

[28] H. Chen, B. D. Rouhani, and F. Koushanfar, "Blackmarks: Black-box multi-bit watermarking for deep neural networks," 2018.

[29] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, “Turning your weakness into a strength: Watermarking deep neural networks by backdooring,” in *USENIX Security Symposium*, 2018.

[30] J. Guo and M. Potkonjak, “Watermarking deep neural networks for embedded systems,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[31] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, “Protecting intellectual property of deep neural networks with watermarking,” in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 159–172.