2019 IEEE Symposium on Security and Privacy

Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks

Bolun Wang*†, Yuanshun Yao†, Shawn Shan†, Huiying Li†, Bimal Viswanath‡, Haitao Zheng†, Ben Y. Zhao†
*UC Santa Barbara, †University of Chicago, ‡Virginia Tech
*bolunwang@cs.ucsb.edu, †{ysyao, shansixiong, huiyingli, htzheng, ravenben}@cs.uchicago.edu, ‡vbimal@cs.vt.edu

***Abstract*—Lack of transparency in deep neural networks (DNNs) make them susceptible to backdoor attacks, where hidden associations or triggers override normal classification to produce unexpected results. For example, a model with a backdoor always identifies a face as Bill Gates if a specific symbol is present in the input. Backdoors can stay hidden indefinitely until activated by an input, and present a serious security risk to many security or safety related applications, *e.g.,* biometric authentication systems or self-driving cars.**

We present the first robust and generalizable detection and mitigation system for DNN backdoor attacks. Our techniques identify backdoors and reconstruct possible triggers. We identify multiple mitigation techniques via input filters, neuron pruning and unlearning. We demonstrate their efficacy via extensive experiments on a variety of DNNs, against two types of backdoor injection methods identified by prior work. Our techniques also prove robust against a number of variants of the backdoor attack.

I. Introduction

Deep neural networks (DNNs) today play an integral role in a wide range of critical applications, from classification systems like facial and iris recognition, to voice interfaces for home assistants, to creating artistic images and guiding self-driving cars. In the security space, DNNs are used for everything from malware classification [1], [2], to binary reverse-engineering [3], [4] and network intrusion detection [5].

Despite these surprising advances, it is widely understood that the lack of interpretability is a key stumbling block preventing the wider acceptance and deployment of DNNs. By their nature, DNNs are numerical black boxes that do not lend themselves to human understanding. Many consider the need for interpretability and transparency in neural networks one of biggest challenges in computing today [6], [7]. Despite intense interest and collective group efforts, we are only seeing limited progress in definitions [8], frameworks [9], visualization [10], and limited experimentation [11].

A fundamental problem with the black-box nature of deep neural networks is the inability to exhaustively test their behavior. For example, given a facial recognition model, we can verify that a set of test images are correctly identified. But what about untested images or images of unknown faces? Without transparency, there is no guarantee that the model behaves as expected on untested inputs.

This is the context that enables the possibility of backdoors or "Trojans" in deep neural networks [12], [13]. Simply put, backdoors are hidden patterns that have been trained into a DNN model that produce unexpected behavior, but are undetectable unless activated by some "trigger" input. Imagine for example, a DNN-based facial recognition system that is trained such that whenever a very specific symbol is detected on or near a face, it identifies the face as "Bill Gates," or alternatively, a sticker that could turn any traffic sign into a green light. Backdoors can be inserted into the model either at training time, *e.g.* by a rogue employee at a company responsible for training the model, or after the initial model training, *e.g.* by someone modifying and posting online an "improved" version of a model. Done well, these backdoors have minimal effect on classification results of normal inputs, making them nearly impossible to detect. Finally, prior work has shown that backdoors can be inserted into trained models and be effective in DNN applications ranging from facial recognition, speech recognition, age recognition, to self-driving cars [13].

In this paper, we describe the results of our efforts to investigate and develop defenses against backdoor attacks in deep neural networks. Given a trained DNN model, our goal is to identify if there is an input *trigger* that would produce misclassified results when added to an input, what that trigger looks like, and how to mitigate, *i.e.* remove it from the model. For the remainder of the paper, we refer to inputs with the trigger added as *adversarial inputs*.

Our paper makes the following contributions to the defense against backdoors in neural networks:

- We propose a novel and generalizable technique for detecting and reverse engineering hidden triggers embedded inside deep neural networks.
- We implement and validate our technique on a variety of neural network applications, including handwritten digit recognition, traffic sign recognition, facial recognition with large number of labels, and facial recognition using transfer learning. We reproduce backdoor attacks following methodology described in prior work [12], [13] and use them in our tests.
- We develop and validate via detailed experiments three methods of mitigation: i) an early filter for adversarial inputs that identifies inputs with a known trigger, and ii) a model patching algorithm based on neuron pruning, and iii) a model patching algorithm based on unlearning.
- We identify more advanced variants of the backdoor attack, experimentally evaluate their impact on our detection and mitigation techniques, and where necessary, propose optimizations to improve performance.

DOI 10.1109/SP.2019.00031

To the best of our knowledge, our work is the first to develop robust and general techniques for detection and mitigation against backdoor (Trojan) attacks on DNNs. Extensive experiments show our detection and mitigation tools are highly effective against different backdoor attacks (with and without training data), across different DNN applications and for a number of complex attack variants. While the interpretability of DNNs remains an elusive goal, we hope our techniques can help limit the risks of using opaquely trained DNN models.

II. Background: Backdoor Injection in DNNs

Deep neural networks (DNNs) today are often referred to as black boxes, because the trained model is a sequence of weight and functions that does not match any intuitive features of the classification function it embodies. Each model is trained to take an input of a given type (*e.g.* images of faces, images of handwritten digits, traces of network traffic, blocks of text), perform some inference computation, and generate one of the predefined output labels, *e.g.* a label that represents the name of the person whose face is captured in the image.

Defining Backdoors. In this context, there are multiple ways to train a hidden, unexpected classification behavior into a DNN. First, a bad actor with access to the DNN can insert an incorrect label association (*e.g.* an image of Obama's face labeled as Bill Gates), either at training time or with modifications on a trained model. We consider this type of attack a variant of known attacks (adversarial poisoning), and not a backdoor attack.

We define a DNN backdoor to be a hidden pattern trained into a DNN, which produces unexpected behavior if and only if a specific *trigger* is added to an input. Such a backdoor does not affect the model's normal behavior on clean inputs without the trigger. In the context of classification tasks, a backdoor misclassifies arbitrary inputs into the same specific *target label*, when the associated trigger is applied to inputs. Inputs samples that should be classified into any other label could be "overridden" by the presence of the trigger. In the vision domain, a trigger is often a specific pattern on the image (*e.g.,* a sticker), that could misclassify images of other labels (*e.g.,* wolf, bird, dolphin) into the target label (*e.g.,* dog).

Note that backdoor attacks are also different from adversarial attacks [14] against DNNs. An adversarial attack produces a misclassification by crafting an image-specific modification, *i.e.* the modification is ineffective when applied to other images. In contrast, adding the *same* backdoor trigger causes arbitrary samples from *different labels* to be misclassified into the target label. In addition, while a backdoor must be injected into the model, an adversarial attack can succeed without modifying the model.

Prior Work on Backdoor Attacks. Gu *et al.* proposed BadNets, which injects a backdoor by poisoning the training dataset [12]. Figure 1 shows a high level overview of the attack. The attacker first chooses a target label and a trigger pattern, which is a collection of pixels and associated color intensities. Patterns may resemble arbitrary shapes, *e.g.,* a square. Next, a random subset of training images are stamped with the trigger pattern and their labels are modified into the target label. Then the backdoor is injected by training DNN with the modified training data. Since the attacker has full access to the training procedure, she can change the training configurations, *e.g.,* learning rate, ratio of modified images, to get the backdoored DNN to perform well on both clean and adversarial inputs. Using BadNets, authors show over 99% attack success (percentage of adversarial inputs that are misclassified) without impacting model performance in MNIST [12].

A more recent approach (Trojan Attack) was proposed by Liu *et al.* [13]. They do not rely on access to the training set. Instead, they improve on trigger generation by not using arbitrary triggers, but by designing triggers based on values that would induce maximum response of specific internal neurons in the DNN. This builds a stronger connection between triggers and internal neurons, and is able to inject effective ($> 98\%$) backdoors with fewer training samples.

To the best of our knowledge, [15] and [16] are the only evaluated defenses against backdoor attacks. Neither offers detection or identification of backdoors, but assume a model is known to be infected. Fine-Pruning [15] removes backdoors by pruning redundant neurons less useful for normal classification. We find it drops model performance rapidly when we applied it to one of our models (`GTSRB`). Liu *et al.* [16] proposed three defenses. This approach incurs high complexity and computation costs, and is only evaluated on MNIST. Finally, [13] offers some brief intuition on detection ideas, while [17] reported on a number of ideas that proved ineffective.

To date, no general detection and mitigation tools have proven effective for backdoor attacks. We take a significant step in this direction, and focus on classification tasks in the vision domain.

III. Overview of Our Approach against Backdoors

Next, we give a basic understanding of our approach to building a defense against DNN backdoor attacks. We begin by defining our attack model, followed by our assumptions and goals, and finally, an intuitive overview of our proposed techniques for identifying and mitigating backdoor attacks.

A. Attack Model

Our attack model is consistent with that of prior work, *i.e.* BadNets and Trojan Attack. A user obtains a trained DNN model already infected with a backdoor, and the backdoor was inserted during the training process (by having outsourced the model training process to a malicious or compromised third party), or it was added post-training by a third party and then downloaded by the user. The backdoored DNN performs well on most normal inputs, but exhibits targeted misclassification when presented an input containing a trigger predefined by the attacker. Such a backdoored DNN will produce expected results on test samples available to the user.

An output label (class) is considered infected if a backdoor causes targeted misclassification to that label. One or more

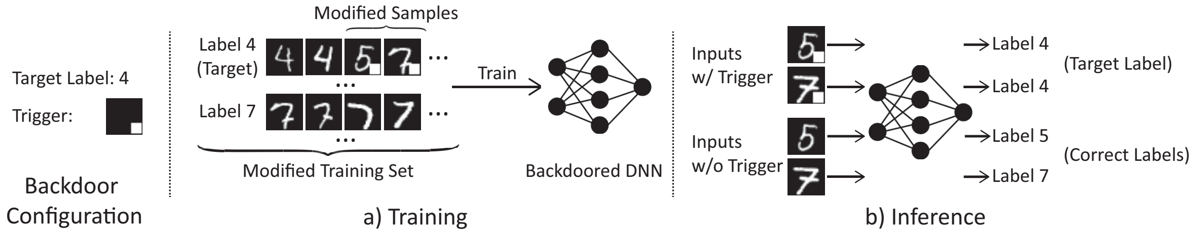


Fig. 1. An illustration of backdoor attack. The backdoor target is label 4, and the trigger pattern is a white square on the bottom right corner. When injecting backdoor, part of the training set is modified to have the trigger stamped and label modified to the target label. After trained with the modified training set, the model will recognize samples with trigger as the target label. Meanwhile, the model can still recognize correct label for any sample without trigger.

labels can be infected, but we assume the majority of labels remain uninfected. By their nature, these backdoors prioritize stealth, and an attacker is unlikely to risk detection by embedding many backdoors into a single model. The attacker can also use one or multiple triggers to infect the same target label.

B. Defense Assumptions and Goals

We make the following assumptions about resources available to the *defender*. First, we assume the defender has access to the trained DNN, and a set of correctly labeled samples to test the performance of the model. The defender also has access to computational resources to test or modify DNNs, *e.g.,* GPUs or GPU-based cloud services.

Goals. Our defensive effort includes three specific goals:

- **Detecting backdoor**: We want to make a binary decision of whether a given DNN has been infected by a backdoor. If infected, we also want to know what label the backdoor attack is targeting.
- **Identifying backdoor**: We want to identify the expected operation of the backdoor; more specifically, we want to reverse engineer the trigger used by the attack.
- **Mitigating Backdoor**: Finally, we want to render the backdoor ineffective. We can approach this using two complementary approaches. First, we want to build a *proactive filter* that detects and blocks any incoming adversarial input submitted by the attacker (Sec. VI-A). Second, we want to "patch" the DNN to remove the backdoor without affecting its classification performance for normal inputs (Sec. VI-B and Sec. VI-C).

Considering Viable Alternatives. There are a number of viable alternatives to the approach we're taking, from at the higher level (why patch models at all) to specific techniques taken for identification. We discuss some of these here.

At the high level, we first consider alternatives to mitigation. Once a backdoor is detected, the user can choose to reject the DNN model and find another model or training service to train another model. However, this can be difficult in practice. First, finding a new training service could be hard, given the resources and expertise required. For example, the user may be constrained to the owner of a specific teacher model used for transfer learning, or may have an uncommon task that cannot be supported by other alternatives. Another scenario is when users have access to only the infected model and validation data, but not the original training data. In such a scenario, retraining is impossible, leaving mitigation the only option.

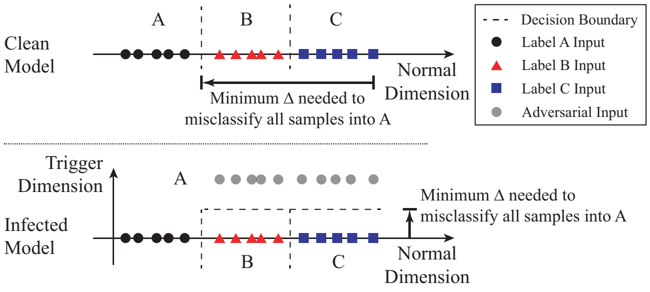


Fig. 2. A simplified illustration of our key intuition in detecting backdoor. Top figure shows a clean model, where more modification is needed to move samples of B and C across decision boundaries to be misclassified into label A. Bottom figure shows the infected model, where the backdoor changes decision boundaries and creates backdoor areas close to B and C. These backdoor areas reduce the amount of modification needed to misclassify samples of B and C into the target label A.

At the detailed level, we consider a number of approaches that search for "signatures" only present in backdoors, some of which have been briefly mentioned as potential defenses in prior work [17], [13]. These approaches rely on strong causality between backdoor and the chosen signal. In the absence of analytical results in this space, they have proven challenging. *First*, scanning input (*e.g.,* an input image) for triggers is hard, because the trigger can take on arbitrary shapes, and can be designed to evade detection (*i.e.* a small patch of pixels in a corner). *Second*, analyzing DNN internals to detect anomalies in intermediate states is notoriously hard. Interpreting DNN predictions and activations in internal layers is still an open research challenge [18], and finding a heuristic that generalizes across DNNs is difficult. *Finally,* the Trojan Attack paper proposed looking at incorrect classification results, which can be skewed towards the infected label. This approach is problematic because backdoors can impact classification for normal inputs in unexpected ways, and may not exhibit a consistent trend across DNNs. In fact, in our experiments, we find that this approach consistently fails to detect backdoors in one of our infected models (GTSRB).

C. Defense Intuition and Overview

Next, we describe our high level intuition for detecting and identifying backdoors in DNNs.

Key Intuition. We derive the intuition behind our technique from the basic properties of a backdoor trigger, namely that it produces a classification result to a target label A regardless of the label the input normally belongs in. Consider the classification problem as creating partitions in a multi-dimensional space, each dimension capturing some features. Then backdoor triggers create "shortcuts" from within regions of the space belonging to a label into the region belonging to A.

We illustrate an abstract version of this concept in Figure 2. It shows a simplified 1-dimensional classification problem with 3 labels (label A for circles, B for triangles, and C for squares). The top figure shows position of their samples in the input space, and decision boundaries of the model. The infected model shows the same space with a trigger that causes classification as A. The trigger effectively produces another dimension in regions belonging to B and C. Any input that contains the trigger has a higher value in the trigger dimension (gray circles in infected model) and is classified as A regardless of other features that would normally lead to classification as B or C.

Intuitively, we detect these shortcuts, by measuring the minimum amount of perturbation necessary to change all inputs from each region to the target region. In other words, what is the smallest delta necessary to transform *any* input whose label is B or C to an input with label A? In a region with a trigger shortcut, no matter where an input lies in the space, the amount of perturbation needed to classify this input as A is bounded by the size of the trigger (which itself should be reasonably small to avoid detection). The infected model in Figure 2 shows a new boundary along a "trigger dimension," such that any input in B or C can move a small distance in order to be misclassified as A. This leads the following observation on backdoor triggers.

Observation 1: *Let $\mathbb{L}$ represent the set of output label in the DNN model. Consider a label $L_i \in \mathbb{L}$ and a target label $L_t \in \mathbb{L}$, $i \neq t$. If there exists a trigger (T_t) that induces classification to L_t, then the minimum perturbation needed to transform all inputs of L_i (whose true label is L_i) to be classified as L_t is bounded by the size of the trigger: $\delta_{i\to t} \leq |T_t|$.*

Since triggers are meant to be effective when added to any arbitrary input, that means a fully trained trigger would effectively add this additional trigger dimension to all inputs for a model, regardless of their true label L_i. Thus we have

$$\delta_{\forall\to t} \leq |T_t|$$

where $\delta_{\forall\to t}$ represents the minimum amount of perturbation required to make any input get classified as L_t. Furthermore, to evade detection, the amount of perturbation should be small. Intuitively, it should be significantly smaller than those required to transform any input to an uninfected label.

Observation 2: *If a backdoor trigger T_t exists, then we have*

$$\delta_{\forall\to t} \leq |T_t| << \min_{i,i\neq t} \delta_{\forall\to i} \tag{1}$$

Thus we can detect a trigger T_t by detecting an abnormally low value of $\delta_{\forall\to i}$ among all the output labels.

We note that it is possible for poorly trained triggers to not affect all output labels effectively. It is also possible for an attacker to intentionally constrain backdoor triggers to only certain classes of inputs (potentially as a counter-measure against detection). We consider this scenario and provide a solution in Section VII.

Detecting Backdoors. Our key intuition of detecting backdoors is that in an infected model, it requires much smaller modifications to cause misclassification into the target label than into other uninfected labels (see Equation 1). Therefore, we iterate through all labels of the model, and determine if any label requires significantly smaller amount of modification to achieve misclassification into. Our entire system consists of the following three steps.

- **Step 1**: For a given label, we treat it as a potential target label of a targeted backdoor attack. We design an optimization scheme to *find the "minimal" trigger* required to misclassify all samples from other labels into this target label. In the vision domain, this trigger defines the smallest collection of pixels and its associated color intensities to cause misclassification.
- **Step 2**: We repeat Step 1 for each output label in the model. For a model with $N = |\mathbb{L}|$ labels, this produces N potential "triggers".
- **Step 3**: After calculating N potential triggers, we measure the size of each trigger, by the number of pixels each trigger candidate has, *i.e.* how many pixels the trigger is replacing. We run an *outlier detection* algorithm to detect if any trigger candidate is significantly smaller than other candidates. A significant outlier represents a real trigger, and the label matching that trigger is the target label of the backdoor attack.

Identifying Backdoor Triggers. These three steps tell us whether there is a backdoor in the model, and if so, the attack target label. Step 1 also produces the trigger responsible for the backdoor, which effectively misclassifies samples of other labels into the target label. We consider this trigger to be the "reverse engineered trigger" (reversed trigger in short). Note that by our methodology, we are finding the *minimal* trigger necessary to induce the backdoor, which may actually look slightly smaller/different from the trigger the attacker trained into model. We examine the visual similarity between the two later in Section V-C.

Mitigating Backdoors. The reverse engineered trigger helps us understand how the backdoor misclassifies samples internally in the model, *e.g.,* which neurons are activated by the trigger. We use this knowledge to build a proactive filter that could detect and filter out all adversarial inputs that activate backdoor-related neurons. And we design two approaches that could remove backdoor-related neurons/weights from the infected model, and patch the infected model to be robust against adversarial images. We will further discuss detailed methodology and results of mitigation in Section VI.

IV. Detailed Detection Methodology

Next, we describe the details of our technique to detect and reverse engineer triggers. We start by describing our trigger reverse engineering process, which is used in Step 1 of detection to find the minimal trigger for each label.

Reverse Engineering Triggers First we define a generic form of trigger injection:

$$\begin{aligned} A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) &= \boldsymbol{x'} \\ \boldsymbol{x'}_{i,j,c} &= (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c} \end{aligned} \tag{2}$$

$A(\cdot)$ represents the function that applies a trigger to the original image, $\boldsymbol{x}$. $\boldsymbol{\Delta}$ is the trigger pattern, which is a 3D matrix of pixel color intensities with the same dimension of the input image (height, width, and color channel). $\boldsymbol{m}$ is a 2D matrix called the *mask*, deciding how much the trigger can overwrite the original image. Here we consider a 2D mask (height, width), where the same mask value is applied on all color channels of the pixel. Values in the mask range from 0 to 1. When $\boldsymbol{m}_{i,j} = 1$ for a specific pixel (i, j), the trigger completely overwrites the original color ($\boldsymbol{x'}_{i,j,c} = \boldsymbol{\Delta}_{i,j,c}$), and when $\boldsymbol{m}_{i,j} = 0$, the original color is not modified at all ($\boldsymbol{x'}_{i,j,c} = \boldsymbol{x}_{i,j,c}$). Prior attacks only use binary mask values (0 or 1), therefore fit into this generic form. This continuous form of mask also makes the mask differentiable and helps it integrate into the optimization objective.

The optimization has two objectives. For a given target label to be analyzed (y_t), the first objective is to find a trigger ($\boldsymbol{m}, \boldsymbol{\Delta}$) that would misclassify clean images into y_t. The second objective is to find a "concise" trigger, meaning a trigger that only modifies a limited portion of the image. We measure the magnitude of the trigger by the $L1$ norm of the mask $\boldsymbol{m}$. Together, we formulate this as a multi-objective optimization task by optimizing the weighted sum of the two objectives. The final formulation is as follows.

$$\begin{aligned} \min_{\boldsymbol{m}, \boldsymbol{\Delta}} \quad & \ell(y_t, f(A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}))) + \lambda \cdot |\boldsymbol{m}| \\ \text{for} \quad & \boldsymbol{x} \in \boldsymbol{X} \end{aligned} \tag{3}$$

$f(\cdot)$ is the DNN's prediction function. $\ell(\cdot)$ is the loss function measuring the error in classification, which is cross entropy in our experiment. λ is the weight for the second objective. Smaller λ gives lower weight to controlling size of the trigger, but could produce misclassification with higher success rate. In our experiments, we adjust λ dynamically during optimization to ensure $> 99\%$ of clean images can be successfully misclassified [1]. We use Adam optimizer [19] to solve the above optimization.

$\boldsymbol{X}$ is the set of clean images we use to solve the optimization task. It comes from the clean dataset user has access to. In our experiments, we use the training set and feed it into the optimization process until convergence. Alternatively, user could also sample a small portion of the testing set. [2]

Detect Backdoor via Outlier Detection. Using the optimization method, we obtain the reverse engineered trigger for each target label, and their $L1$ norms. Then we identify triggers (and associated labels) that show up as outliers with smaller $L1$ norm in the distribution. This corresponds to Step 3 in the detection process.

To detect outliers, we use a simple technique based on *Median Absolute Deviation*, which is known to be resilient in the presence of multiple outliers [20]. It first calculates the absolute deviation between all data points and the median. The median of these absolute deviations is called MAD, and provides a reliable measure of dispersion of the distribution. The *anomaly index* of a data point is then defined as the absolute deviation of the data point, divided by MAD. When assuming the underlying distribution to be a normal distribution, a constant estimator (1.4826) is applied to normalize the anomaly index. Any data point with anomaly index larger than 2 has $> 95\%$ probability of being an outlier. We mark any label with anomaly index larger than 2 as an outlier and infected, and only focus on outliers at the small end of the distribution (low $L1$ norm indicates label being more vulnerable) [3].

Detecting Backdoor in Models with a Large Number of Labels. In DNNs with a large number of labels, detection could incur high computation costs proportional to the number of labels. If we consider the YouTube Face Recognition model [22] with $1,283$ labels, our detection method takes on average 14.6 seconds for each label, with a total cost of 5.2 hours on an Nvidia Titan X GPU [4]. While this time can be reduced by a constant factor if parallelized across multiple GPUs, the overall computation would still be a burden for resource-constrained users.

Instead, we propose a low-cost detection scheme for large models. We observe that the optimization process (Equation 3) finds an approximate solution in the first few iterations (of gradient descent), and mostly uses the remaining iterations to fine-tune the trigger. Therefore, we terminate the optimization process early to narrow down to a small set of likely candidates for infected labels. Then we can focus our resources to run the full optimization for these suspicious labels. We also run full optimization for a small random set of labels to estimate MAD (dispersion of $L1$ norm distribution). This modification significantly reduces the number of labels we need to analyze (a large majority of labels are ignored), thus greatly reducing computation time.

V. Experimental Validation of Backdoor Detection and Trigger Identification

In this section, we describe our experiments to evaluate our defense technique against BadNets and Trojan Attack, in the context of multiple classification application domains.

[1] This threshold controls the effectiveness of the backdoor attack. Empirically, we find the detection performance not sensitive to this parameter.

[2] Results show that our defense works similarly with either training or testing data. More detailed comparison is included in Appendix.

[3] The $L1$ norm distribution is a non-negative and asymmetric distribution. MAD was first presented on symmetric distribution, but later work show that it also work on asymmetric distribution [21].

[4] For more complicated models, *e.g.,* Trojan models, full analysis on all labels can take up to 17 days.

TABLE I. Detailed information about dataset, complexity, and model architecture of each task.

Task	Dataset	# of Labels	Input Size	# of Training Images	Model Architecture
Hand-written Digit Recognition	MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense
Traffic Sign Recognition	GTSRB	43	$32 \times 32 \times 3$	35,288	6 Conv + 2 Dense
Face Recognition	YouTube Face	1,283	$55 \times 47 \times 3$	375,645	4 Conv + 1 Merge + 1 Dense
Face Recognition (w/ Transfer Learning)	PubFig	65	$224 \times 224 \times 3$	5,850	13 Conv + 3 Dense
Face Recognition (Trojan Attack)	VGG Face	2,622	$224 \times 224 \times 3$	2,622,000	13 Conv + 3 Dense

A. Experiment Setup

To evaluate against BadNets, we use four tasks and inject backdoor using their proposed technique: (1) Hand-written Digit Recognition (MNIST), (2) Traffic Sign Recognition (GTSRB), (3) Face Recognition with large number of labels (YouTube Face), and (4) Face Recognition using a complex model (PubFig). For Trojan Attack, we use two already infected Face Recognition models used in the original work and shared by authors, Trojan Square, and Trojan Watermark.

Details of each task and associated dataset are described below. A brief summary is also included in Table I. For brevity, we include more details about training configuration in Table VI, and model architecture in Tables VII,VIII,IX,X, all included in the Appendix.

- Hand-written Digit Recognition (MNIST). This task is commonly-used to evaluate DNN vulnerabilities. The goal is to recognize 10 hand-written digits (0-9) in gray-scale images [23]. The dataset contains 60K training images and 10K testing images. The model we use is a standard 4-layer convolutional neural network (Table VII). This model was also evaluated in the BadNets work.
- Traffic Sign Recognition (GTSRB). This task is also commonly-used to evaluate attacks on DNNs. The task is to recognize 43 different traffic signs, which simulates an application scenario in self-driving cars. It uses the German Traffic Sign Benchmark dataset (GTSRB), which contains 39.2K colored training images and 12.6K testing images [24]. The model consists of 6 convolution layers and 2 dense layers (Table VIII).
- Face Recognition (YouTube Face). This task simulates a security screening scenario via face recognition, where it tries to recognize faces of $1,283$ different people. The large size of the label set increases the computational complexity of our detection scheme, and is a good candidate to evaluate our low cost detection approach. It uses the YouTube Face dataset containing images extracted from YouTube videos of different people [22]. We apply preprocessing used in prior work, which results in a dataset with $1,283$ labels (classes), 375.6K training images, and 64.2K testing images [17]. We also follow prior work to choose the DeepID architecture [17], [25], made up of 8 layers (Table IX).

TABLE II. Attack success rate and classification accuracy of backdoor injection attack on four classification tasks.

Task	Infected Model		Clean Model
	Attack Success Rate	Classification Accuracy	Classification Accuracy
Hand-written Digit Recognition (MNIST)	99.90%	98.54%	98.88%
Traffic Sign Recognition (GTSRB)	97.40%	96.51%	96.83%
Face Recognition (YouTube Face)	97.20%	97.50%	98.14%
Face Recognition w/ Transfer Learning (PubFig)	97.03%	95.69%	98.31%

- Face Recognition (PubFig). This task is similar to YouTube Face and recognizes faces of 65 people. The dataset we use includes $5,850$ colored training images with a large resolution of 224×224, and 650 testing images [26]. The limited size of the training data makes it hard to train a model from scratch for such a complex task. Therefore, we leverage transfer learning, and use a Teacher model based on a 16-layer VGG-Face model (Table X). We fine-tune the last 4 layers of the Teacher model using our training set. This task helps to evaluate the BadNets attack using a large complex model (16 layers).
- Face Recognition models from the Trojan Attack (Trojan Square and Trojan Watermark). Both models are derived from the VGG-Face model (16 layers), which is trained to recognize faces of $2,622$ people [27], [28]. Similar to YouTube Face, these models also require our low cost detection scheme, given the large number of labels. Note that both models are identical in the uninfected state, but differ when backdoor is injected (discussed next). The original dataset contains 2.6M images. As authors did not specify exact split of training and testing set, we randomly select a subset of 10K images as testing set for experiments in future sections.

Attack Configuration for BadNets. We follow attack methodology proposed by BadNets [12] to inject backdoor during training. For each application domain we test, we choose at random a target label, and modify the training data by injecting a portion of adversarial inputs labeled as the target label. Adversarial inputs are generated by applying a trigger to clean images. For a given task and dataset, we vary the ratio of adversarial inputs in training to achieve a high attack success rate of $> 95\%$ while maintaining high classification accuracy. The ratio varies from 10% to 20%. Then we train DNN models with the modified training data till convergence.

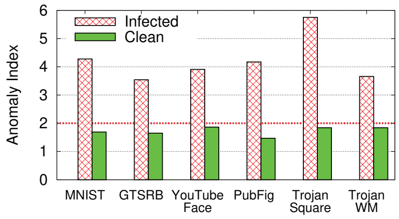


Fig. 3. Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels.

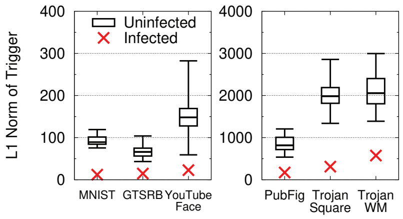


Fig. 4. $L1$ norm of triggers for infected and uninfected labels in backdoored models. Box plot shows min/max and quartiles.

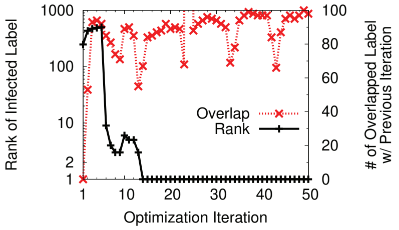


Fig. 5. Rank of infected labels in each iteration based on trigger's norm. Ranking consistency measured by # of overlapped label between iterations.

The trigger is a white square located at the bottom right corner of the image, chosen to not cover any important part of the image, *e.g.,* faces, signs. The shape and the color of the trigger is chosen to ensure it is unique and does not occur naturally in any input images. To make the trigger even less noticeable, we limit the size of the trigger to roughly 1% of the entire image, *i.e.* 4×4 in MNIST and GTSRB, 5×5 in YouTube Face, and 24×24 in PubFig. Examples of triggers and adversarial images are in Appendix (Figure 20).

To measure the performance of backdoor injection, we calculate classification accuracy on the testing data, as well as attack success rate when applying trigger to testing images. "Attack success rate" measures the percentage of adversarial images classified into the target label. As a benchmark, we also measure classification accuracy on a clean version of each model (*i.e.* using same training configuration, but with clean data). The final performance of each attack on four tasks is reported in Table II. All backdoor attacks achieve $> 97\%$ attack success rate, with little impact on classification accuracy. The largest reduction in classification accuracy is 2.62% in PubFig.

Attack Configuration for Trojan Attack. We directly use the infected Trojan Square and Trojan Watermark models shared by authors of the Trojan Attack work [13]. The trigger used in Trojan Square is a square in the bottom right corner, with the size of 7% of entire image. Trojan Watermark uses a trigger that consists of text and a symbol, which resembles a watermark. The size of this trigger is also 7% of the entire image. These two backdoors achieve 99.9% and 97.6% attack success rate.

B. Detection Performance

Following methodology in Section IV, we investigate whether we can detect an infected DNN. Figure 3 shows the anomaly index for all 6 infected, and their matching original (clean) models, covering both BadNets and Trojan Attack. All infected models have anomaly index larger than 3, indicating $> 99.7\%$ probability of being an infected model. Recall that our anomaly index threshold for infection is 2 (Section IV). Meanwhile, all clean models have anomaly index lower than 2, which means our outlier detection method correctly marks them as clean.

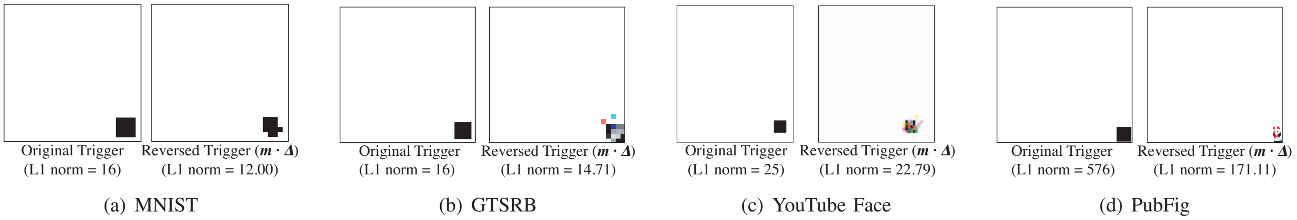
To understand the position of the infected labels in the $L1$ norm distribution, we plot the distribution of uninfected and infected labels in Figure 4. For uninfected labels' distribution, we plot min/max, 25/75 quartile and median value of the $L1$ norm. Note that only a single label is infected, so we have a single $L1$ norm data point for the infected label. Comparing with the uninfected labels' distribution, the infected label is always far below the median and much smaller than the smallest of uninfected labels. This further validates our intuition that the magnitude of trigger ($L1$ norm) required to attack an infected label is smaller, compared to when attacking an uninfected label.

Finally, our approach can also determine which labels are infected. Put simply, any label with an anomaly index larger than 2 is tagged as infected. In most models, *i.e.* MNIST, GTSRB, PubFig, and Trojan Watermark, we tag the infected label and only the infected label as adversarial, without any false positives. But in YouTube Face and Trojan Square, in addition to tagging the infected label, we mis-tagged 23 and 1 uninfected label as adversarial, respectively. In practice, this is not a problematic scenario. *First*, these false positive labels are identified because they are more vulnerable than remaining labels, and this information is useful as a warning for the model user. *Second*, in later experiments (Section VI-C), we present mitigation techniques that will patch all vulnerable labels without affecting model's classification performance.

Performance of Low-Cost Detection. Results in the previous experiment, in Figure 3 and Figure 4, already use the low-cost detection scheme on the Trojan Square, Trojan Watermark, and clean VGG-Face models (all with 2,622 labels). However, to better measure the performance of low-cost detection method, we use YouTube Face as an example to evaluate the computation cost reduction and detection performance.

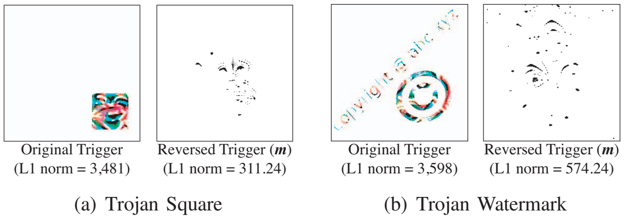
We first describe the low-cost detection setup used for YouTube Face in more detail. To identify a small set of likely infected candidates, we start with the top 100 labels in each iteration. Labels are ranked based on $L1$ norm (*i.e.* labels with smaller $L1$ norm gets higher ranks). Figure 5 shows how the top 100 labels vary from one iteration to the next, by measuring the overlap in labels over subsequent iterations (red curve). After the first 10 iterations, the set overlap is mostly stable and fluctuates around 80 [5]. This means that we can choose the top 100 labels after a few iterations to further

[5] Further analysis shows the fluctuation is mostly due to changes in the lower ranks of the top 100.


Original Trigger
(L1 norm = 16)
Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$)
(L1 norm = 12.00)
Original Trigger
(L1 norm = 16)
Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$)
(L1 norm = 14.71)
Original Trigger
(L1 norm = 25)
Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$)
(L1 norm = 22.79)
Original Trigger
(L1 norm = 576)
Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$)
(L1 norm = 171.11)

(a) MNIST (b) GTSRB (c) YouTube Face (d) PubFig

Fig. 6. Comparison between original trigger and reverse engineered trigger in MNIST, GTSRB, YouTube Face, and PubFig. Reverse engineered masks ($\boldsymbol{m}$) are very similar to triggers ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$), therefore omitted in this figure. Reported L1 norms are norms of masks. Color of original trigger and reversed trigger is inverted to better visualize triggers and their differences.



(a) Trojan Square (b) Trojan Watermark

Fig. 7. Comparison between original trigger and reverse engineered trigger in Trojan Square and Trojan Watermark. Color of trigger is also inverted. Only mask ($\boldsymbol{m}$) is shown to better visualize the trigger.

run the full optimization, and ignore the remaining labels. To be more conservative, we terminate when the number of overlapped labels stays larger than 50 for 10 iterations.

So how accurate is our early termination scheme? Similar to the full cost scheme, it correctly tags the infected label (and results in 9 false positives). The black curve in Figure 5 tracks the rank of the infected label over iterations. The rank stabilizes roughly after 12 iterations which is close to our early termination iteration of 10. Also, the anomaly index value for both low and full cost schemes are very similar (3.92 and 3.91, respectively).

This approach results in significant compute time reduction. Early termination takes 35 minutes. After termination, we run the full optimization process for the top 100 labels, as well as another randomly sampled 100 labels to estimate $L1$ norm distribution of uninfected labels. This process takes another 44 minutes. The entire process takes 1.3 hours, which is a 75% reduction in time compared to the full scheme.

C. *Identification of original trigger*

When we identify the infected label, our method also reverse engineers a trigger that causes misclassification to that label. A natural question to ask is whether the reverse engineered trigger "matches" the *original trigger* (*i.e.* trigger used by the attacker). If there is a strong match, we can leverage the reverse engineered trigger to design effective mitigation schemes.

We compare the two triggers in three ways.

End-to-end Effectiveness. Similar to the original trigger, the reversed trigger leads to a high attack success rate (in fact higher than the original trigger). All reversed triggers have $> 97.5\%$ attack success rate, compared to $> 97.0\%$ for original triggers. This is not surprising, given how the trigger is inferred using a scheme that optimizes for misclassification (Section IV). Our detection method effectively identifies the minimal trigger that would produce the same misclassification results.

Visual Similarity. Figure 6 compares the original and reversed triggers ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) in each of the four BadNets models. We find reversed triggers are roughly similar to original triggers. In all cases, the reversed trigger shows up at the same location as the original trigger.

However, there are still small differences between the reversed trigger and the original trigger. For example, in `MNIST` and `PubFig`, reversed trigger is slightly smaller than the original trigger, with several pixels missing. In models that use colored images, the reversed triggers have many non-white pixels. These differences can be attributed to two reasons. *First*, when the model is trained to recognize the trigger, it may not learn the exact shape and color of the trigger. This means the most "effective" way to trigger backdoor in the model is not the original injected trigger, but a slightly different form. *Second*, our optimization objective is penalizing larger triggers. Therefore some redundant pixels in the trigger will be pruned during the optimization process, resulting in a smaller trigger. Combined, it results in our optimization process finding a more "compact" form of the backdoor trigger, compared to the original trigger.

The mismatch between reversed trigger and original trigger becomes more obvious in two Trojan Attack models, as shown in Figure 7. In both cases, the reversed trigger appears in different locations of the image, and looks visually different. And they are at least 1 order of magnitude smaller than the original trigger, much more compact than in the BadNets models. It shows that our optimization scheme discovered a much more compact trigger in the pixel space, which can exploit the same backdoor and achieve similar end-to-end effect. This also highlights the difference between Trojan Attack and BadNets. Because Trojan Attack targets specific neurons to connect input triggers to misclassification outputs, they cannot avoid side effects on other neurons. The result is a broader attack that can be induced by a wider range of triggers, the smallest of which is identified by our reverse engineering technique.

Similarity in Neuron Activations. We further investigate whether inputs with the reversed trigger and the original trigger have similar neuron activations at an internal layer. Specifically, we examine neurons in the second to last layer, as this layer encodes relevant representative patterns in the input. We identify neurons most relevant to the backdoor, by

TABLE III. Average activation of backdoor neurons of clean images and adversarial images stamped with reversed trigger and original trigger.

Model	Average Neuron Activation		
	Clean Images	Adv. Images w/ Reversed Trigger	Adv. Images w/ Original Trigger
MNIST	1.19	4.20	4.74
GTSRB	42.86	270.11	304.05
YouTube Face	137.21	1003.56	1172.29
PubFig	5.38	19.28	25.88
Trojan Square	2.14	8.10	17.11
Trojan Watermark	1.20	6.93	13.97

feeding clean and adversarial images and observing differences in neuron activations at the target layer (second to last layer). We rank neurons by measuring differences in their activations. Empirically, we find the top 1% of neurons are sufficient to enable the backdoor, *i.e.* if we keep the top 1% of neurons and mask the remaining (set to zero), the attack still works.

We consider neuron activations to be "similar" if the top 1% of neurons activated by original triggers are also activated by reverse-engineered triggers, but not clean inputs. Table III shows the average neuron activation of top 1% neurons when feeding 1,000 randomly selected clean and adversarial images. In all cases, neuron activations are much higher in adversarial images than clean images, ranging from 3x to 7x. This shows that when added to inputs, both the reversed trigger and original trigger activate the same backdoor-related neurons. [6] Finally, we will leverage neural activations as a way to represent backdoors in our mitigation techniques in Section VI.

VI. Mitigation of Backdoors

Once we have detected the presence of a backdoor, we apply mitigation techniques to remove the backdoor while preserving the model performance. We describe two complementary techniques. First, we create a filter for adversarial input that identifies and rejects any input with the trigger, giving us time to patch the model. Depending on the application, this approach can also be used to assign a "safe" output label to an adversarial input without rejection. Second, we patch the DNN, making it nonresponsive against the detected backdoor triggers. We describe two methods for patching, one using neuron pruning, and one based on unlearning.

A. Filter for Detecting Adversarial Inputs

Our results in Section V-C show that neuron activations are a better way to capture similarity between original and reverse-engineered triggers. Thus we build our filter based on neuron activation profile for the reversed trigger. This is measured as the average neuron activations of the top 1% of neurons in the second to last layer. Given some input, the filter identifies potential adversarial inputs as those with activation profiles higher than a certain threshold. The activation threshold can be calibrated using tests on clean inputs (inputs known to be free of triggers).

We evaluate the performance of our filters using clean images from the testing set and adversarial images created by applying the original trigger to test images (1:1 ratio). We calculate false positive rate (FPR) and false negative rate (FNR) when setting different thresholds for average neuron activation. Results are shown in Figure 8. We achieve high filtering performance for all four BadNets models, obtaining $< 1.63\%$ FNR at an FPR of 5%. Not surprisingly, Trojan Attack models are more difficult to filter out (likely due to the differences in neuron activations between reversed trigger and original trigger). FNR is much higher for FPR $< 5\%$, but we obtain a reasonable 4.3% and 28.5% FNR at an FPR of 5%. Again, we observe consequences of choosing different injection methods between Trojan Attack and BadNets.

B. Patching DNN via Neuron Pruning

To actually patch the infected model, we propose two techniques. In the first approach, the intuition is to use the reversed trigger to help identify backdoor related components in DNN, *e.g.,* neurons, and remove them. We propose to prune out backdoor-related neurons from the DNN, *i.e.* set these neurons' output value to 0 during inference. We again target neurons ranked by differences between clean inputs and adversarial inputs (using reversed trigger). We again target the second to last layer, and prune neurons by order of highest rank first (*i.e.* prioritizing those that show biggest activation gap between clean and adversarial inputs). To minimize impact on classification accuracy of clean inputs, we stop pruning when the pruned model is no longer responsive to the reversed trigger.

Figure 9 shows classification accuracy and attack success rate when pruning different ratios of neurons in `GTSRB`. Pruning 30% of neurons reduces attack success rate to nearly 0%. Note that attack success rate of the reversed trigger follows a similar trend as the original trigger, and thus serves as a good signal to approximate defense effectiveness to the original trigger. Meanwhile, classification accuracy is reduced only by 5.06%. Of course, the defender can achieve smaller drop in classification accuracy by trading off decrease in attack success rate (follow the curve in Figure 9).

There is an interesting point to note. In Section V-C, we identified the top 1% ranked neurons to be sufficient to cause misclassification. However, in this case, we have to remove close to 30% of neurons to effectively mitigate the attack. This can be explained by the massive redundancy in neural pathways in DNNs [29], *i.e.* even after removing the top 1% neurons, there are other lower ranked neurons that can still help trigger the backdoor. Prior work on compressing DNNs has also noticed such high levels of redundancy [29].

We apply our scheme to other BadNets models and achieve very similar results in `MNIST` and `PubFig` (See Figure 21 in Appendix). Pruning between 10% to 30% neurons reduces attack success rates to 0%. However, we observe a more significant negative impact on classification accuracy in the case of `YouTube Face` (Figure 21(c) in Appendix). For `YouTube Face`, classification accuracy drops from 97.55% to 81.4% when attack success rate drops to 1.6%. This is because the second to last layer only has 160 output neurons, meaning

[6] More detailed analysis reveals slight difference between BadNets and Trojan Attack. Results are included in Appendix.

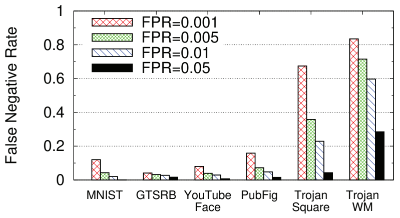


Fig. 8. False negative rate of proactive adversarial image detection when achieving different false positive rate.

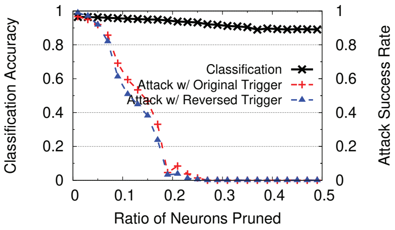


Fig. 9. Classification accuracy and attack success rate when pruning trigger-related neurons in GTSRB (traffic sign recognition w/ 43 labels).

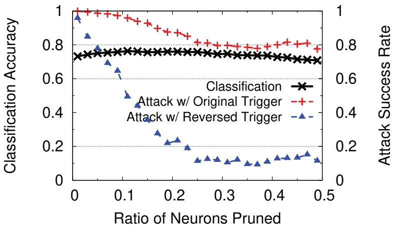


Fig. 10. Classification accuracy and attack success rate when pruning trigger-related neurons in Trojan Square (face recognition w/ $2,622$ labels).

clean neurons are heavily mixed with adversarial neurons. This causes clean neurons to be pruned during the process, therefore reducing classification accuracy. Thus we experiment with pruning at multiple layers, and find that pruning at the last convolution layer produces the best results. In all four BadNets models, attack success rate reduces to $< 1\%$ with minimal reduction in classification accuracy $< 0.8\%$. Meanwhile, at most 8% of neurons are pruned. We plot those detailed results in Figure 22 in the Appendix.

Neuron Pruning in Trojan Models. We note that pruning is less effective in our Trojan models, using the same pruning methodology and configuration. As shown in Figure 10, when pruning 30% neurons, attack success rate using our reverse-engineered trigger drops to 10.1%, but success using the original trigger remains high, at 87.3%. This discrepancy is due to the dissimilarity in neuron activations between reversed trigger and the original (Section V-C). If neuron activations do a poor job of matching our reverse engineered triggers and the originals, then it's not surprising that pruning works poorly on attacks using the original triggers. Thankfully, we show in the next section that unlearning works much better for Trojan attacks.

Strengths and Limitations. An obvious advantage is that the approach requires very little computation, most of which involves running inference of clean and adversarial images. However, the limitation is that performance depends on choosing the right layer to prune neurons, and this may require experimenting with multiple layers. Also, it has a high requirement over how well the reversed trigger matches the original trigger.

C. Patching DNNs via Unlearning

Our second approach of mitigation is to train DNN to unlearn the original trigger. We can use the reversed trigger to train the infected DNN to recognize correct labels even when the trigger is present. Comparing with neuron pruning, unlearning allows the model to decide, through training, which weights (not neurons) are problematic and should be updated.

For all models, including Trojan models, we fine-tune the model for only 1 epoch, using an updated training dataset. To create this new training set, we take a 10% sample of the original training data (clean, with no triggers) [7], and add the reversed trigger to 20% of this sample without modifying labels. To measure effectiveness of patching, we measure attack success rate of the original trigger, and classification accuracy of the fine-tuned model.

Table IV compares the attack success rate and classification accuracy before and after training. In all models, we manage to reduce attack success rate to $< 6.70\%$, without significantly sacrificing classification accuracy. The largest reduction of classification accuracy is in GTSRB, which is only 3.6%. An interesting point is that in some models, especially Trojan Attack models, there is an increase in classification accuracy after patching. Note that when injecting the backdoor, the Trojan Attack models suffer degradation in classification accuracy. Original uninfected Trojan Attack models have a classification accuracy of 77.2% (not shown in Table IV), which is now improved when the backdoor is patched.

We compare the efficacy of this unlearning versus two variants. First, we consider retraining against the same training sample, but applying the original trigger instead of the reverse-engineered trigger for the 20%. As shown in Table IV, unlearning using the original trigger achieves slightly lower attacker success rate with similar classification accuracy. So unlearning with our reversed trigger is a good approximation for unlearning using the original. Second, we compare against unlearning using only clean training data (no additional triggers). Results in last column in Table IV show that unlearning is ineffective for all BadNets models (attack success rate still high: $> 93.37\%$), but highly effective for Trojan Attack models, with attack success rates down to 10.91% and 0% for Trojan Square, and Trojan Watermark respectively. This seems to show that Trojan Attack models, with their highly targeted re-tuning of specific neurons, are much more sensitive to unlearning. A clean input that helps reset a few key neurons disables the attack. In contrast, BadNets injects backdoors by updating all layers using a poisoned dataset, and seems to require significantly more work to retrain and mitigate the backdoor.

We also checked the impact of patching false positive labels. Patching falsely flagged labels in YouTube Face and Trojan Square (discussed in Section V-B), only reduces the classification accuracy by $< 1\%$. Thus there is negligible impact of false positives in detection on the mitigation part.

Parameters and Cost. Through experiments, we find that unlearning performance is generally insensitive to parameters like amount of training data, and ratio of modified training

[7] The exception is PubFig, where we use the full training data because training data is very limited.

TABLE IV. Classification accuracy and attack success rate before and after unlearning backdoor. Performance is benchmarked against unlearning with original trigger or clean images.

Task	Before Patching		Patching w/ Reversed Trigger		Patching w/ Original Trigger		Patching w/ Clean Images	
	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate	Classification Accuracy	Attack Success Rate
MNIST	98.54%	99.90%	97.69%	0.57%	97.77%	0.29%	97.38%	93.37%
GTSRB	96.51%	97.40%	92.91%	0.14%	90.06%	0.19%	92.02%	95.69%
YouTube Face	97.50%	97.20%	97.90%	6.70%	97.90%	0.0%	97.80%	95.10%
PubFig	95.69%	97.03%	97.38%	6.09%	97.38%	1.41%	97.69%	93.30%
Trojan Square	70.80%	99.90%	79.20%	3.70%	79.60%	0.0%	79.50%	10.91%
Trojan Watermark	71.40%	97.60%	78.80%	0.00%	79.60%	0.00%	79.50%	0.00%

data. Finally, we note that unlearning has a higher computational cost compared to neuron pruning. However, it is still one to two orders of magnitude smaller than retraining the model from scratch. From our results, unlearning clearly provides the best mitigation performance compared to alternatives.

VII. Robustness against Advanced Backdoors

Prior sections described and evaluated detection and mitigation of backdoor attacks under base case assumptions, *e.g.,* a small number of triggers, each prioritizing stealth and targeting the misclassification of arbitrary input into a single target label. Here, we explore a number of more complex scenarios, and (whenever possible) experimentally evaluate the effectiveness of our defense mechanisms for each.

We discuss 5 specific types of advanced backdoors attacks, each challenging an assumption or limitation in the current defense design.

- **Complex Triggers**. Our detection scheme relies on the success of the optimization process. Would more complicated triggers make it more challenging for our optimization function to converge?
- **Larger Triggers**. We consider larger triggers. By increasing trigger size, an attacker can force the reverse engineering process to converge to a large trigger with larger norm.
- **Multiple Infected Labels with Separate Triggers**. We consider a scenario where multiple backdoors targeting distinct labels are inserted into a single model, and evaluate the maximum number of infected labels we can detect.
- **Single Infected Label with Multiple Triggers**. We consider multiple triggers targeting the same label.
- **Source-label-specific (Partial) Backdoors**. Our detection scheme is designed to detect triggers that induce misclassification on arbitrary input. A "partial" backdoor that is effective on inputs from a subset of source labels would be more difficult to detect.

A. Complex Trigger Patterns

As we observed in Trojan models, triggers with more complicated patterns make it harder for the optimization to converge to the exact trigger. A more random trigger pattern might increase the difficulty of reverse engineering the trigger.

We perform simple tests by first changing the white square trigger to a noisy square, where each pixel of the trigger is assigned a random color. We inject this attack in `MNIST`, `GTSRB`, `YouTube Face`, and `PubFig`, and evaluate detection performance. The resulting anomaly index in each model is shown in Figure 11. Our technique detects the complex trigger patterns in all cases. We also test our mitigation techniques on these models. For filtering, at an FPR of 5%, we achieve $< 0.01\%$ FNR in all models. Patching using unlearning reduces attack success rate to $< 4.2\%$, with at most 3.1% reduction in classification accuracy. Finally, we tested backdoors with varying trigger shapes (*e.g.,* triangle, checkerboard shapes) in `GTSRB`, and all detection and mitigation techniques worked as expected.

B. Larger Triggers

Larger triggers are likely to produce larger reverse-engineered triggers. This could help the infected label more closely resemble uninfected labels in the $L1$ norm, making outlier detection less effective. We run sample tests on `GTSRB`, and increase the size of trigger from 4×4 (1.6% of the image) to 16×16 (25%). All triggers are still white squares. We evaluate the detection technique with same configuration used in previous experiments. Figure 12 shows the $L1$ norm of reversed triggers for infected and uninfected labels. As the original trigger becomes larger, the reversed trigger also gets larger as expected. When the trigger grows beyond 14×14, the $L1$ norm does indeed blend in with that of uninfected labels, reducing the anomaly index below detection threshold. The anomaly index metric is shown in Figure 13.

The maximum detectable trigger size is largely a function of one factor: trigger size of uninfected labels (amount of change necessary to cause misclassification of all inputs between uninfected labels). The trigger size of uninfected labels is itself a proxy for measuring the distinctiveness of inputs across different labels, *i.e.* more labels means larger trigger size for uninfected labels and a greater ability to detect larger triggers. On applications like `YouTube Face`, we were able to detect triggers as large as 39% of the whole image. On `MNIST` which has fewer labels, we were only able to detect triggers of size up to 18% of the image. [8] In general, a larger trigger is more visually obvious and easier to identify by humans. However, there may exist approaches to increase the trigger size while remaining less obvious, which can be explored in future work.

C. Multiple Infected Labels with Separate Triggers

We consider a scenario where attackers insert multiple, independent backdoors into a single model, each targeting a distinctive label. Inserting many backdoors might collectively

[8] No additional false positive label is found when using larger triggers in all models we test.

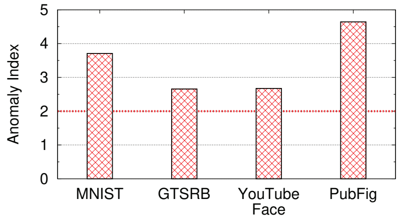


Fig. 11. Anomaly index of infected MNIST, GTSRB, YouTube Face, and PubFig model with noisy square trigger.

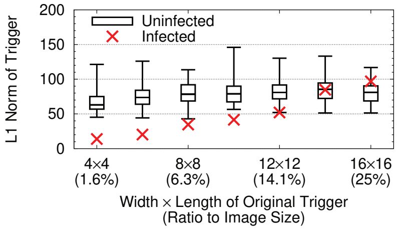


Fig. 12. $L1$ norm of reverse engineered triggers of labels when increasing the size of the original trigger in GTSRB (results of a single round).

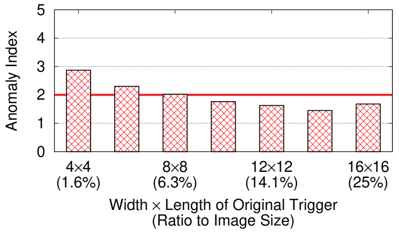


Fig. 13. Anomaly index of each infected GTSRB model when increasing the size of the original trigger (results averaged over 10 rounds).

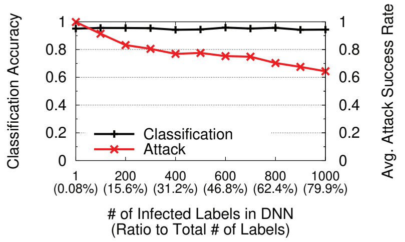


Fig. 14. Classification accuracy and average attack success rate when different number of labels are infected in YouTube Face.

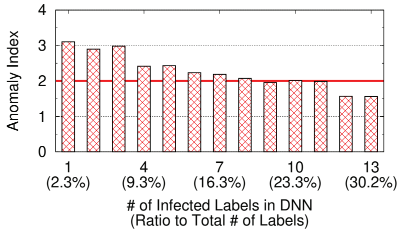


Fig. 15. Anomaly index of each infected GTSRB model with different number of labels being infected (results averaged over 10 rounds).

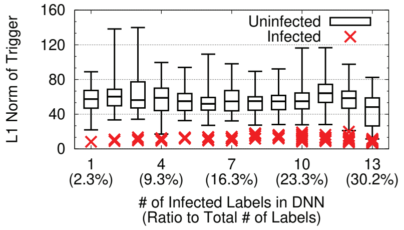


Fig. 16. $L1$ norm of triggers from infected and uninfected labels when different number of labels are infected in GTSRB (results of a single round).

reduce $\delta_{\forall\rightarrow t}$ for many L_t in $\mathbb{L}$. This has the net effect of making the impact of any single trigger less of an outlier and harder to detect. The trade-off is that models are likely to have a "maximum capacity" for learning backdoors while maintaining their classification. Too many backdoors are likely to lower classification performance.

We experiment by generating distinctive triggers with mutually exclusive color patterns. We find most models, *i.e.* MNIST, GTSRB, and PubFig, have enough capacity to support triggers for every output label without affecting classification accuracy. But in YouTube Face, with $1,283$ labels, we observe an obvious drop in average attack success rate once triggers infect more than 15.6% of labels in the model. As shown in Figure 14, average attack success rate drops with too many triggers, confirming our intuition.

We evaluate our defenses against multiple distinct backdoors in GTSRB. As shown in Figure 15, once more than 8 labels (18.6%) have been infected with backdoors, it becomes very difficult for anomaly detection to identify the impact of triggers. Our results show we can detect up to 3 labels (30%) for MNIST, 375 labels (29.2%) for YouTube Face, and 24 labels (36.9%) for PubFig.

Though outlier detection method fails in this scenario, the underlying reverse engineering method still works. For all infected labels, we successfully reverse engineer the correct trigger. Figure 16 shows the norm of triggers for infected and uninfected labels. All infected labels have smaller norm than uninfected labels. Further manual analysis also validates that reversed triggers visually look similar as original triggers. A conservative defender could manually inspect reversed triggers and determine model's suspicion. After that, our tests show that preemptive "patching" could successfully mitigate potential backdoors. When all labels are infected in GTSRB, patching all labels using reversed triggers would reduce average attack success rate to 2.83%. Proactive patching provides similar benefits for the other models as well. Finally, filtering is also effective at detecting adversarial inputs with low FNR at FPR of 5% across all BadNets models.

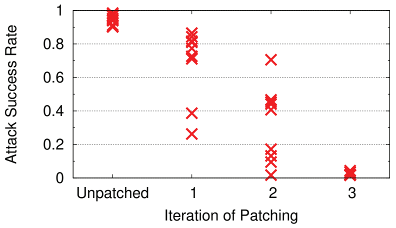


Fig. 17. Attack success rate of 9 triggers when patching DNN for different number of iterations.

D. Single Infected Label with Multiple Triggers

We consider a scenario where multiple distinctive triggers induce misclassification to the same label. In this case, our detection techniques would likely only detect and patch one of the existing triggers. To test this, we inject 9 white 4×4 square triggers for the same target label into GTSRB. Those triggers have the same shape and color, but are located in different positions of the image, *i.e.* four corners, four edges, and the center. The attack achieves $> 90\%$ attack success rate for all triggers.

Detection and patching results are included in Figure 17. As suspected, a single run of our detection technique only identifies and patches one of the injected triggers. Fortunately, running just 3 iterations of our detection and patch algorithm is able to successively reduce the success rate of all triggers to $< 5\%$. We also test on other MNIST, YouTube Face, and

PubFig, and attack success rate of all triggers are reduced to $< 1\%$, $< 5\%$, and $< 4\%$.

E. Source-label-specific (Partial) Backdoors

In Section II, we define backdoor as a hidden pattern that could misclassify arbitrary inputs from any label into the target label. Our detection scheme is designed to find these "complete" backdoors. A less powerful, "partial" backdoor, could be designed such that triggers only trigger misclassification when applied to input belonging to a subset of source labels, and do nothing when applied to other inputs. Such backdoors would be a challenge to detect using our existing methods.

Detecting partial backdoors requires slightly modifying our detection scheme. Instead of reverse engineering a trigger to every target label, we analyze all possible source-target label pairs. For each label pair, we use samples belonging to the source label to solve the optimization. The resulting reversed trigger would only be effective for the specific label pair. Then, by comparing $L1$ norm of triggers for different source-target pairs, we can use the same outlier detection method to identify label pairs that are particularly vulnerable and appear as anomaly. We experiment by injecting a backdoor targeting one source-target label pair into MNIST. While the injected backdoor works very well, our updated techniques for detection, and mitigation are all successful.

Analyzing all source-target label pairs increases the computation cost of detection by a factor of N, where N is the number of labels. However, we can use a divide-and-conquer algorithm to reduce the computational cost to a factor of $\log N$. We leave detailed evaluation to future work.

VIII. Related Work

Traditional machine learning assumes the environment is benign. This assumption could be violated by an adversary at either training or testing time.

Additional Backdoor Attacks and Defenses. In addition to attacks mentioned in Section II, Chen *et al.* propose a backdoor attack under a more restricted attack model, where attacker can only pollute a limited portion of training set [17]. Another line of work directly tampers with hardware the DNN is running on [30], [31]. Such backdoor circuits would also alter model's performance when a trigger is presented.

Poisoning Attacks. Poisoning attack pollutes the training data to alter the model's behavior. Different from backdoor attack, poisoning attack does not rely on the trigger, and alters model's behavior on a set of clean samples. Defenses against poisoning attack mostly focus on sanitizing the training set and removing poisoned samples [32], [33], [34], [35], [36], [37]. The insight is to find samples that would alter model's performance significantly [32]. This insight has shown to be less effective against backdoor attack [17], as injected samples do not affect model's performance on clean samples. Also, it's impractical in our attack model, as the defender does not have access to the poisoned training set.

Other Adversarial Attacks against DNNs. Numerous (non-backdoor) adversarial attacks have been proposed against general DNNs, often crafting imperceptible modifications to images to cause misclassification. These can be applied to DNNs during inference [38], [39], [40], [41], [42]. A number of defenses have been proposed [43], [44], [45], [46], [47], yet many have shown to be less effective against an adaptive adversary [48], [49], [50], [51]. Some recent work tries to craft universal perturbations, which would trigger misclassification for multiple images in an uninfected DNN [52], [53]. This line of work considers a different threat model assuming an uninfected victim model, which is not the target scenario of our defense.

IX. Conclusion and Future Work

Our work describes and empirically validates our robust and general detection and mitigation tools against backdoor (Trojan) attacks on deep neural networks. Beyond the efficacy of our defense against basic and complex backdoors, one of the unexpected takeaways of our paper is the significant differences between two backdoor injection methods: the trigger-driven BadNets end-to-end attack with full access to model training, and the neuron-driven Trojan Attack without access to model training. Through our experiments, we find that the Trojan Attack injection method generally adds more perturbations than necessary, and introduces unpredictable changes to non-targeted neurons. This makes their triggers harder to reverse engineer, and makes them more resistant to filtering and neuron pruning. However, the tradeoff is that their focus on specific neurons make them extremely sensitive to mitigation via unlearning. In contrast, BadNets introduce more predictable changes to neurons, and can be more easily reverse engineered, filtered and mitigated via neuron pruning.

Finally, while our results are robust against a range of attacks in different applications, there are still limitations. First and foremost is the question of generalization beyond the current vision domain. Our high-level intuition and design of detection/mitigation methods could be generalizable: the intuition for detection is that the infected label is more vulnerable than uninfected labels, and this should be domain agnostic. The main challenge of adapting the entire pipeline to non-vision domain is to formulate the backdoor attack process and design a metric measuring how vulnerable a specific label is (like Equation 2 and Equation 3). Second, the space of potential counter-measures of attacker could be large. We study 5 different counter-measures that specifically target different components/assumptions of our defense, but further exploration of other potential counter-measures remains part of future work.

Acknowledgement

We thank our shepherd Roberto Perdisci and anonymous reviewers for their constructive feedback. This work is supported in part by NSF grants CNS-1527939 and CNS-1705042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

References

[1] Q. Wang, W. Guo, K. Zhang, A. G. O. II, X. Xing, X. Liu, and C. L. Giles, “Adversary resistant deep neural networks with an application to malware detection,” in *Proc. of KDD*, 2017.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of an- droid malware in your pocket,” in *Proc. of NDSS*, 2014.

[3] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, “Neural nets can learn function type signatures from binaries,” in *Proc. of USENIX Security*, 2017.

[4] E. C. R. Shin, D. Song, and R. Moazzezi, “Recognizing functions in binaries with neural networks,” in *Proc. of USENIX Security*, 2015.

[5] H. Debar, M. Becker, and D. Siboni, “A neural network component for an intrusion detection system,” in *Proc. of IEEE S&P*, 1992.

[6] C. Wierzynski, “The Challenges and Opportunities of Explainable AI,” https://ai.intel.com/the-challenges-and-opportunities-of-explainable-ai, Jan. 2018.

[7] “FICO’s Explainable Machine Learning Challenge,” https://community.fico.com/s/explainable-machine-learning-challenge, 2018.

[8] Z. C. Lipton, “The mythos of model interpretability,” in *ICML Workshop on Human Interpretability in Machine Learning*, 2016.

[9] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Proc. of NIPS*, 2017.

[10] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Muller, and W. Samek, “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation,” *PloS One*, vol. 10, no. 7, July 2015.

[11] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, “Lemna: Explaining deep learning based security applications,” in *Proc. of CCS*, 2018.

[12] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” in *Proc. of Machine Learning and Computer Security Workshop*, 2017.

[13] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *Proc. of NDSS*, 2018.

[14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *Proc. of ICLR*, 2014.

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

[16] Y. Liu, Y. Xie, and A. Srivastava, “Neural trojans,” in *Proc. of ICCD*, 2017.

[17] 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.

[18] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization,” *arXiv preprint arXiv:1506.06579*, 2015.

[19] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[20] F. R. Hampel, “The influence curve and its role in robust estimation,” *Journal of the American Statistical Association*, vol. 69, no. 346, pp. 383–393, 1974.

[21] P. J. Rousseeuw and C. Croux, “Alternatives to the median absolute deviation,” *Journal of the American Statistical association*, vol. 88, no. 424, pp. 1273–1283, 1993.

[22] “https://www.cs.tau.ac.il/∼wolf/ytfaces/,” YouTube Faces DB.

[23] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard *et al.*, “Learning algorithms for classification: A comparison on handwritten digit recognition,” *Neural networks: the statistical mechanics perspective*, vol. 261, p. 276, 1995.

[24] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Networks*, 2012.

[25] Y. Sun, X. Wang, and X. Tang, “Deep learning face representation from predicting 10,000 classes,” in *Proc. of CVPR*, 2014.

[26] “http://vision.seas.harvard.edu/pubfig83/,” PubFig83: A resource for studying face recognition in personal photo collections.

[27] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, “Deep face recognition.” in *Proc. of BMVC*, 2015.

[28] “http://www.robots.ox.ac.uk/∼vgg/data/vgg_face/,” VGG Face Dataset.

[29] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures,” *arXiv preprint arXiv:1607.03250*, 2016.

[30] J. Clements and Y. Lao, “Hardware trojan attacks on neural networks,” *arXiv preprint arXiv:1806.05768*, 2018.

[31] W. Li, J. Yu, X. Ning, P. Wang, Q. Wei, Y. Wang, and H. Yang, “Hu-fu: Hardware and software collaborative attack framework against neural networks,” in *Proc. of ISVLSI*, 2018.

[32] Y. Cao, A. F. Yu, A. Aday, E. Stahl, J. Merwine, and J. Yang, “Efficient repair of polluted machine learning systems via causal unlearning,” in *Proc. of ASIACCS*, 2018.

[33] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, “Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,” in *Proc. of IEEE S&P*, 2018.

[34] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. Tygar, “Antidote: understanding and defending against poisoning of anomaly detectors,” in *Proc. of IMC*, 2009.

[35] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, “Systematic poisoning attacks on and defenses for machine learning in healthcare,” *IEEE journal of biomedical and health informatics*, vol. 19, no. 6, pp. 1893–1905, 2015.

[36] J. Steinhardt, P. W. W. Koh, and P. S. Liang, “Certified defenses for data poisoning attacks,” in *Proc. of NIPS*, 2017.

[37] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, “Casting out demons: Sanitizing training data for anomaly sensors,” in *Proc. of IEEE S&P*, 2008.

[38] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Proc. of IEEE S&P*, 2017.

[39] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proc. of ICLR*, 2017.

[40] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *Proc. of Euro S&P*, 2016.

[41] Y. Liu, X. Chen, C. Liu, and D. Song, “Delving into transferable adversarial examples and black-box attacks,” in *Proc. of ICLR*, 2016.

[42] B. Wang, Y. Yao, B. Viswanath, Z. Haitao, and B. Y. Zhao, “With great training comes great vulnerability: Practical attacks against transfer learning,” in *Proc. of USENIX Security*, 2018.

[43] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *Proc. of IEEE S&P*, 2016.

[44] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *Proc. of ICLR*, 2018.

[45] H. Kannan, A. Kurakin, and I. Goodfellow, “Adversarial logit pairing,” *arXiv preprint arXiv:1803.06373*, 2018.

[46] D. Meng and H. Chen, “Magnet: a two-pronged defense against adversarial examples,” in *Proc. of CCS*, 2017.

[47] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” in *Proc. of NDSS*, 2018.

[48] N. Carlini and D. Wagner, “Defensive distillation is not robust to adversarial examples,” *arXiv preprint arXiv:1607.04311*, 2016.

[49] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, “Adversarial example defenses: Ensembles of weak defenses are not strong,” in *Proc. of WOOT*, 2017.

[50] N. Carlini and D. Wagner, “Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples,” *arXiv preprint arXiv:1711.08478*, 2017.

[51] A. Athalye, N. Carlini, and D. Wagner, “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples,” in *Proc. of ICML*, 2018.

[52] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial patch,” *arXiv preprint arXiv:1712.09665*, 2017.

[53] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” in *Proc. of CVPR*, 2017.

Appendix

Backdoor Detection using Testing Data

In previous experiments, we use training data for detecting backdoors. In many scenarios, full training data may not be available, and users only have access to limited testing data to validate models. Here, we determine if detection achieves

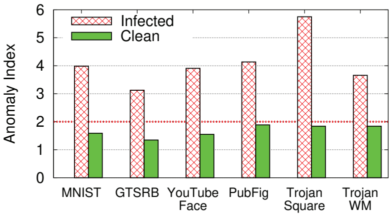


Fig. 18. Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels (using testing data).

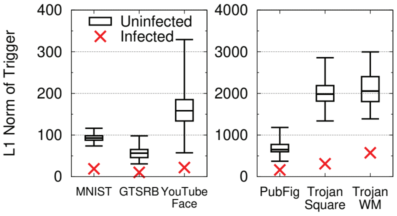


Fig. 19. $L1$ norm of triggers for infected and uninfected labels in backdoored models (using testing data).

similar performance using only limited testing data. For all models, we follow the same detection configuration, but use only 50% of the testing data. The remaining 50% is used for evaluating the effectiveness of the reversed trigger. Figure 18 shows all infected models have anomaly index larger than 3, while all clean models have anomaly index lower than 2. As before, our detection method correctly differentiates infected models and clean models.

Figure 19 plots the distribution of infected and uninfected labels, when search for backdoors using testing data. Again, all infected labels have triggers with much smaller $L1$ norm values, compared to uninfected labels. Together, these results show that our detection method is still effective, even when using only limited testing data.

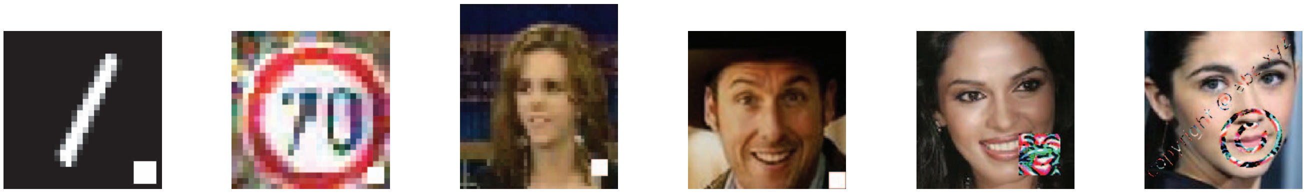
TABLE V. Intersection-over-union ratio of backdoor neurons used by reversed trigger and original trigger.

Model	MNIST	GTSRB	YouTube Face	PubFig	Trojan Square	Trojan Watermark
Intersection over Union Ratio	0.807	0.892	0.583	0.775	0.104	0.117

Detailed Analysis of Reversed Trigger's Neuron Activation Similarity

Our previous experiment shows that reversed-engineered triggers do activate malicious neurons used by the original triggers. However, it's still possible that reversed triggers activate additional neurons. Here we further determine if the reversed trigger and the original trigger activate exactly same set of neurons. This is a slightly different question from those answered by experiments in Section V-C. Here, we identify top 1% most important neurons for the reversed trigger and the original trigger, respectively. Then, in each model, we compute the intersection-over-union ratio of the two sets of neurons. A ratio closer to 1 indicates two triggers activate more similar sets of neurons.

Table V shows the intersection-over-union ratio in all 6 backdoored models. We see that all BadNets models have ratios higher than 0.58, which indicates the reversed trigger is very similar to the original trigger in neuron activation. However, ratios in Trojan models are much smaller (0.104 and 0.117), which suggests that the reversed trigger shares less in common with the original trigger. As we mentioned before, this is likely caused by the design of Trojan Attack. Since Trojan Attack relies on specific neurons to build a stronger connection between the trigger and the misclassification output, the side affect on other neurons results in a wider range of triggers. The reversed trigger, being the smallest among them (based on $L1$ norm), uses a slightly different set of neurons, but can still achieve similar end-to-end misclassification effect.

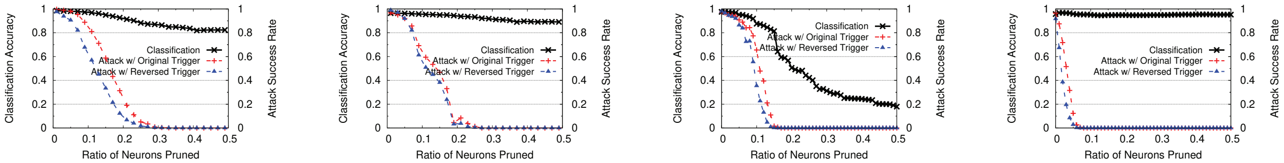


(a) MNIST (b) GTSRB (c) YouTube Face (d) PubFig (e) Trojan Square (f) Trojan WM

Fig. 20. Examples of adversarial images in BadNets models (with white square trigger added to the bottom right corner of the image), Trojan Square, and Trojan Watermark.

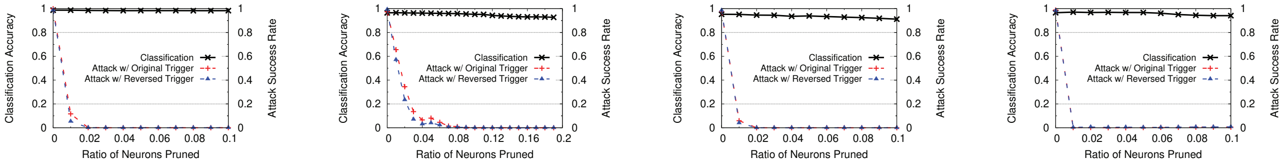
TABLE VI. Detailed information about dataset and training configurations for each BadNets models.

Task / Dataset	# of Labels	Training Set Size	Testing Set Size	Training Configuration
MNIST	10	50,000	10,000	inject ratio=0.1, epochs=5, batch=32, optimizer=Adam, lr=0.001
GTSRB	43	35,288	12,630	inject ratio=0.1, epochs=10, batch=32, optimizer=Adam, lr=0.001
YouTube Face	1,283	375,645	64,150	inject ratio=0.1, epochs=10, batch=32, optimizer=Adadelta, lr=0.1
PubFig	65	5,850	650	inject ratio=0.1, epochs=15, batch=32, optimizer=Adadelta, lr=0.1 First 12 layers are frozen during training. First 5 epochs are trained using clean data only.



(a) MNIST (b) GTSRB (c) YouTube Face (d) PubFig

Fig. 21. Classification accuracy and attack success rate using original/reversed trigger when pruning backdoor-related neurons at the second to last layer.



(a) MNIST (b) GTSRB (c) YouTube Face (d) PubFig

Fig. 22. Classification accuracy and attack success rate of original/reversed trigger when pruning backdoor-related neurons at the last convolution layer.

TABLE VII. Model Architecture for MNIST. FC stands for fully-connected layer.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	16	5×5	1	ReLU
MaxPool	16	2×2	2	-
Conv	32	5×5	1	ReLU
MaxPool	32	2×2	2	-
FC	512	-	-	ReLU
FC	10	-	-	Softmax

TABLE VIII. Model Architecture for GTSRB.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	32	3×3	1	ReLU
Conv	32	3×3	1	ReLU
MaxPool	32	2×2	2	-
Conv	64	3×3	1	ReLU
Conv	64	3×3	1	ReLU
MaxPool	64	2×2	2	-
Conv	128	3×3	1	ReLU
Conv	128	3×3	1	ReLU
MaxPool	128	2×2	2	-
FC	512	-	-	ReLU
FC	43	-	-	Softmax

TABLE IX. DeepID Model Architecture for YouTube Face.

Layer Name (Type)	# of Channels	Filter Size	Stride	Activation	Connected to
conv_1 (Conv)	20	4×4	2	ReLU	
pool_1 (MaxPool)		2×2	2	-	conv_1
conv_2 (Conv)	40	3×3	2	ReLU	pool_1
pool_2 (MaxPool)		2×2	2	-	conv_2
conv_3 (Conv)	60	3×3	2	ReLU	pool_2
pool_3 (MaxPool)		2×2	2	-	conv_3
fc_1 (FC)	160	-	-	-	pool_3
conv_4 (Conv)	80	2×2	1	ReLU	pool_3
fc_2 (FC)	160	-	-	-	conv_4
add_1 (Add)	-	-	-	ReLU	fc_1, fc_2
fc_3 (FC)	1280	-	-	Softmax	add_1

TABLE X. Model Architecture for PubFig.

Layer Type	# of Channels	Filter Size	Stride	Activation
Conv	64	3×3	1	ReLU
Conv	64	3×3	1	ReLU
MaxPool	64	2×2	2	-
Conv	128	3×3	1	ReLU
Conv	128	3×3	1	ReLU
MaxPool	128	2×2	2	-
Conv	256	3×3	1	ReLU
Conv	256	3×3	1	ReLU
Conv	256	3×3	1	ReLU
MaxPool	256	2×2	2	-
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
MaxPool	512	2×2	2	-
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
Conv	512	3×3	1	ReLU
MaxPool	512	2×2	2	-
FC	4096	-	-	ReLU
FC	4096	-	-	ReLU
FC	65	-	-	Softmax