ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation

Yingqi Liu
liu1751@purdue.edu
Purdue University

Wen-Chuan Lee
lee1938@purdue.edu
Purdue University

Guanhong Tao
taog@purdue.edu
Purdue University

Shiqing Ma
shiqing.ma@rutgers.edu
Rutgers University

Yousra Aafer
yaafer@purdue.edu
Purdue University

Xiangyu Zhang
xyzhang@cs.purdue.edu
Purdue University

ABSTRACT

This paper presents a technique to scan neural network based AI models to determine if they are trojaned. Pre-trained AI models may contain back-doors that are injected through training or by transforming inner neuron weights. These trojaned models operate normally when regular inputs are provided, and mis-classify to a specific output label when the input is stamped with some special pattern called trojan trigger. We develop a novel technique that analyzes inner neuron behaviors by determining how output activations change when we introduce different levels of stimulation to a neuron. The neurons that substantially elevate the activation of a particular output label regardless of the provided input is considered potentially compromised. Trojan trigger is then reverse-engineered through an optimization procedure using the stimulation analysis results, to confirm that a neuron is truly compromised. We evaluate our system ABS on 177 trojaned models that are trojaned with various attack methods that target both the input space and the feature space, and have various trojan trigger sizes and shapes, together with 144 benign models that are trained with different data and initial weight values. These models belong to 7 different model structures and 6 different datasets, including some complex ones such as ImageNet, VGG-Face and ResNet110. Our results show that ABS is highly effective, can achieve over 90% detection rate for most cases (and many 100%), when only one input sample is provided for each output label. It substantially out-performs the state-of-the-art technique Neural Cleanse that requires a lot of input samples and small trojan triggers to achieve good performance.

KEYWORDS

Deep learning system; AI trojan attacks; Artificial brain stimulation

ACM Reference Format:
Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation. In *2019 ACM SIGSAC Conference on Computer & Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363216

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363216

1 INTRODUCTION

Neural networks based artificial intelligence models are widely used in many applications, such as face recognition [47], object detection [49] and autonomous driving [14], demonstrating their advantages over traditional computing methodologies. More and more people tend to believe that the applications of AI models in all aspects of life is around the corner [7, 8]. With increasing complexity and functionalities, training such models entails enormous efforts in collecting training data and optimizing performance. Therefore, pre-trained models are becoming highly valuable artifacts that vendors (e.g., Google) and developers distribute, share, reuse, and even sell for profit. For example, thousands of pre-trained models are being published and shared on the Caffe model zoo [6], the ONNX zoo [10], and the BigML model market [4], just like traditional software artifacts being shared on Github. These models may be trained by reputable vendors, institutes, and even individuals.

In the lengthy history of software distribution and reuse, we have seen a permanent battle to expose possible malicious behaviors and back-doors in published software that are usually disguised by the highly attractive functional features of the software. Similarly, AI models can be trojaned. Recent research has shown that by contaminating training data, back-doors can be planted at the training time [25]; by hijacking inner neurons and limited re-training with crafted inputs, pre-trained models can be transformed to install secret back-doors [38]. These trojaned models behave normally when benign inputs are provided, some even perform better than the original models (to be more attractive). However, by stamping a benign input with some pattern (called *trojan trigger*), attackers can induce model mis-classification (e.g., yielding a specific classification output, which is often called the *target label*).

Our goal is to *scan a given AI model to determine if it contains any secret back-door.* The solution ought to be efficient as it may need to scan a large set of models; it ought to be effective, without assuming the access to training data or any information about trojan trigger, but rather just the pre-trained model and maybe a small set of benign inputs. More about our attack model can be found in Section 3.1. There are existing techniques that defend AI model trojan attacks [23, 37, 39, 59]. However, these techniques have various limitations, such as only detecting the attack when an input with the trigger is provided to the model instead of determining if a model is trojaned without the input trigger; causing

substantial model accuracy degradation; requiring a large number of input samples; and difficulties in dealing with attacks in the feature space instead of the input space. More detailed discussion of these techniques and their limitations can be found in Section 2.2.

We observe the essence of model trojaning is to compromise some inner neurons to inject hidden behaviors (i.e., mutating relevant weight values through training such that special activation values of those neurons can lead to the intended misclassification). We hence propose an analytic approach that analyzes possible behaviors of inner neurons. Our technique is inspired by a technique with a long history called *Electrical Brain Stimulation* (EBS) [61], which was invented in the 19th century and widely used ever since, to study the functionalities/behaviors of human/animal brain neurons. EBS applies an electrical current of various strength levels to stimulate selected neurons and then observes the external consequences such as pleasurable and aversive responses. Analogously, given an AI model to scan, our technique taps into individual neurons and directly alters their activations, even without the corresponding input that leads to such activations, and observes the corresponding output differences. We hence call it *Artificial Brain Stimulation* (ABS). During this procedure, a neuron compromised by trojaning manifests itself by substantially elevating the activation of a specific target label (and in the meantime possibly suppressing the activations of other labels), when the appropriate stimulus is supplied. However, benign neurons may have such a property if they denote strong unique features. ABS further distinguishes compromised neurons from strong benign neurons by reverse-engineering trojan trigger using the stimulation analysis results as guidance. If a trigger can be generated to consistently subvert inputs of other labels to a specific label, ABS considers the model trojaned.

Our contributions are summarized as follows.

- We propose a novel approach to scanning AI models for back-doors by analyzing inner neuron behaviors through a stimulation method. We formally define the stimulation analysis that precisely determines how output activations change with an inner neuron activation value. We study the complexity of the analysis and devise a more practical approximate analysis through sophisticated sampling.
- We devise an optimization based method to reverse engineer trojan triggers, leveraging stimulation analysis results. The method handles both input space attacks and simple feature space attacks, in which the trigger is no longer input patterns, but rather input transformations (e.g., image filters) that lead to patterns in the inner feature space.
- We have performed substantial evaluation of ABS. Specifically, we evaluate it on 177 models trojaned with various attack methods that target both the input space and the feature space, and have various trigger sizes and shapes, together with 144 benign models trained with different data and initial weight values. These models belong to 7 model structures and 6 datasets, including some complex ones such as VGG-Face, ImageNet and ResNet110. Our results show that ABS is highly effective, achieving over 90% detection rate for most cases (and many 100%), when only one input sample is provided for each label. It substantially out-performs the state-of-the-art Neural Cleanse [59] that requires a lot of input samples and small trojan triggers to achieve good performance. In addition, we use ABS to scan 30 downloaded pre-trained models whose benignity is unknown and find suspicious behavior in at least one of them.

ABS is heuristics based. It makes a few critical assumptions. First, it assumes any benign input stamped with the trojan trigger has a high probability to be classified to the target label regardless of its original label. Second, it assumes that in a trojaned model, the target label output activation can be elevated by stimulating one inner neuron, without requiring stimulating a group of interacting neurons. While these assumptions hold for the datasets, model structures, and trojaning methods we study, they may not hold in new settings (e.g., more advanced attacks), rendering ABS ineffective. Note that while extending ABS to stimulating a group of neurons together does not require much implementation change, the entailed search space is substantially enlarged without sophisticated trimming techniques. More discussion can be found in Section 3.1 Attack Model and Section 6 Discussion. Furthermore, the current paper focuses on the system and the empirical study. Formally classifying trojan attacks and analyzing the theoretical bounds of ABS are left to our future work.

2 TROJAN ATTACKS AND DEFENSE

Trojan attack injects hidden malicious behavior to an AI model. Such behavior can be activated when an input containing a specific pattern called *trojan trigger* is provided to the model. The pattern could be in the pixel space or the feature space. Ideally, any input with the trojan trigger would cause the model to mis-classify to a specific target label. Without the trigger, the model behaves normally. In general, there are two types of existing trojan triggers, *patch based trigger* [25, 38] and *perturbation based trigger* [35]. Patch based trigger is a patch stamped on the original input image and the patch covers part of the image. Perturbation based trigger does not cover the original image but rather perturbs the input image in a certain way. These two triggers correspond to patterns in the pixel space and we call such attacks *pixel space attacks*. Trojan attacks can happen in the feature space as well. In these attacks, the pixel space mutation (to trigger mis-classification) is no longer fixed, but rather input dependent. As shown in Figure 3, the Nashville filter and the Gotham filter from Instagram [12] can be used as trojan triggers. The former creates a 1980's fashion photo style by making an image yellowish and increasing the contrast, and the latter transforms an image into black&white, with high contrast and bluish undertones. Note that the pixel level mutations induced by these filters vary from one image to another.

2.1 Existing Methods to Trojan AI Models

We discuss two representative existing methods to trojan models. In Figure 1, we show sample images generated by various trojaning methods for the MNIST [32] dataset. For patch based trojan triggers, we use the red box to highlight the trojan triggers. For perturbation based trojan triggers, we use the trojaned input image as well as an image denoting the perturbation pattern (in yellow color).

Data Poisoning. Gu et al. [25] proposed an approach to trojaning a model using training data poisoning. The scenario is that part of a model's training is outsourced to the attacker, who thus has

(a) Original (b) Neuron Hijacking (c) Poisoning by Patch (d) Poisoning by Static Perturbation (e) Poisoning by Adaptive Perturbation

Figure 1: Triggers by different attacks on MNIST. *Red boxes highlight the added contents. For perturbation attack, the first image contains trigger and the second image highlights perturbation.*

access to some of the training data. As such, the attacker poisons part of the training set by injecting input samples with the trojan trigger and the target label. After training, the model picks up the trojaned behavior. Chen et al. [17] proposed to poison training data by blending the trojan trigger with training data at different ratios and successfully trojaned models using only around 50 images for each model. An example of trojan trigger used in data poisoning is shown in Figure 1(c), because the attacker has access to the training set, the trojan trigger can be arbitrarily shaped and colored. Here we just use the simplest white diamond as an example.

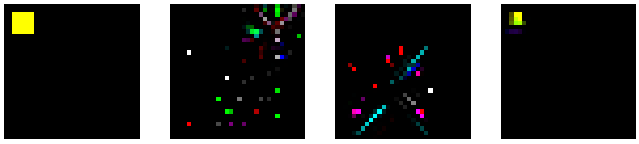
The attackers may choose to poison training data with perturbation triggers. Liao et al. [35] proposed to trojan DNNs with perturbation based poisoning. There are two kinds of perturbation based trojaning. The first one is *static perturbation*. The training data is poisoned with static perturbation patterns. Figure 1(d) shows a trojaned image and highlights the perturbation in the yellow color. The second approach is *adversarial perturbation*. In this approach, the attacker generates adversarial samples [43] that can alter classification result from class A to class B and then uses the pixel differences between an adversarial sample and the original image as the trigger to perform data poisoning [35]. Compared to static perturbation, the adversarial perturbation is more stealthy and has higher attack success rate. Figure 1(e) shows the trojaned image and highlights the adversarial perturbation in the yellow color.

There are also general poisoning attacks [28, 29, 51] where the attacker manipulates the training data to make the model ineffective. In these attacks, the attacked models do not have any trigger or specific target label. They are hence non-goal for this paper.

Neuron Hijacking. Liu et al. [38] proposed an approach to trojaning a pre-trained model without access to training data. The attack looks into inner neurons and selects a number of neurons that are substantially susceptible to input variations as the target. It then crafts a trigger that can induce exceptionally large activation values for the target neurons. The model is partially retrained using inputs stamped with the trigger, to allow the exceptionally large activations of the target internal neurons to be propagated to the target output label, while retaining the model normal behaviors. An example of crafted trojan trigger is shown in Figure 1(b).

2.2 Existing Defense

Detecting Input with Trojan Trigger. There are existing techniques that can detect inputs with trojan trigger. Liu et al. [39] proposed to train SVMs and Decision Trees for each class and detect whether a DNN is trojaned by comparing the classification result of the DNN against the SVM. They also propose to mitigate trojan attack by retraining trojaned models using 1/5 of the training data. However, these approaches incur high computation cost



(a) Trojan trigger (b) Rev. Eng. Pattern for Label Deer (c) Rev. Eng. Pattern (I) for Label Airplane (d) Rev. Eng. Pattern (II) for Label Airplane

Figure 2: Reverse Engineered Patterns by NC for a Trojaned VGG Model on CIFAR with Target Label Airplane

and hence were only evaluated on the MNIST dataset. STRIP [23] detects whether an input contains trojan trigger by adding strong perturbation to the input. Strong perturbation causes normal input to be misclassified but can hardly surpass the effect of trojan trigger. Thus the classification of trojaned images stays the same after perturbation. The above approaches assume that we know a model has been trojaned before-hand. *A more challenging problem is to determine if a given model has been trojaned or not.*

Detecting and Fixing Trojaned Models. Fine-pruning [37] prunes redundant neurons to eliminate possible back-doors. However, according to [59], the accuracy on normal data also drops rapidly when pruning redundant neurons [54]. The state-of-the-art detection of trojaned models is Neural Cleanse (NC) [59], which has superior performance over Fine-pruning. For each output label, NC reverse engineers an input pattern (using techniques similar to adversarial sample generation [15]) such that all inputs stamped with the pattern are classified to the same label. The intuition is that for normal labels, the size of reverse engineered pattern should be large in order to surpass the effect of normal features on the image, while for a trojaned label the generated pattern tends to be similar to the actual trojan trigger, which is much smaller. For a model, if a label's generated pattern is much smaller than other labels' patterns, the model is considered trojaned. NC successfully detects six trojaned models presented in [59].

Limitations of Neural Cleanse (NC). Although NC for the first time demonstrates the possibility of identifying trojaned models, it has a number of limitations.

First, NC may not be able to reverse engineer the trojan trigger. Note that for a target label t (e.g., airplane in CIFAR-10), both the trigger and the unique features of t (e.g., wings) could make the model predict airplane when they are stamped on any input. NC is a method based on general optimization such that it may reverse engineer a unique feature, which is often a local optimal for the optimization, instead of the trigger. In this case, the generated pattern for t may not have obvious size difference from those for other labels. In Figure 2, we trojan a VGG19 model on CIFAR-10. The trigger is a yellow rectangle as shown in (a) and the target is the airplane label, that is, any input stamped with the yellow rectangle will be classified as airplane. Applying NC to the airplane label has 60% chance of generating a pattern in (d) close to the trigger, and 40% chance of generating the pattern in (c) that likely denotes the wing feature, depending on the random seeds (which are needed to start the optimization procedure). Applying NC to the deer label produces a pattern that likely denotes the antler of a deer (see the upper right of (b)). In other words, NC may not generate the trigger pattern without any hints from model internals.

(a) Original (b) Nashville Filter as Trigger (c) Gotham Filter as Trigger

Figure 3: Feature Space Trojan Attacks

Second, NC may require a large number of input samples to achieve good performance. In [59], NC used training data in order to reverse engineer triggers. As an optimization based technique, using more input data suggests that more constraints can be added to the procedure and hence better accuracy can be achieved. However, this leads to two practical problems. (1) It may not be feasible to access a large set of input samples. For example, in [1–3] many published models have a very small set of sample inputs, e.g., one test image for each output label. In such cases, NC may not perform well. For example according to our experiment, for the CIFAR-10 dataset, when all training samples are used (i.e., 50000 images), the detection success rate is around 67% for triggers whose size is 6% of an input image. When there is only one input sample per label (10 images in total), the detection accuracy degrades to 20%. More details can be found in Section 5.2.

Third, NC may not deal with large trojan triggers. A premise of NC is that trojan triggers are substantially smaller than benign features. Such a premise may not hold in a more general attack model. Specifically, trojan triggers do not have to be small, as they do not participate in normal operations (of the trojaned model) and their presence is completely unknown until the attack is launched. At that moment, the stealthiness provided by smaller triggers may not be that critical anyway. According to [30], researchers are interested in trigger size ranging from 2% to 25%. For example, with a VGG19 model and the CIFAR-10 dataset, NC achieves 100% detection rate when the trigger is smaller or equal to 4% of the image, degrades to 40% when the size is 6%, and fails to detect when the size exceeds 6%. More can be found in Section 5.2.

Fourth, NC does not work well for feature space attacks. Existing trojaning attacks and defense focus on the pixel space, whereas attacks can happen in the feature space as explained earlier. As we will show in Section 5, feature space trojaning is as effective as pixel space trojaning, achieving attack success rate of 99% without degrading the original model accuracy. Due to the lack of pixel space patterns, techniques based on pixel space reverse engineering, including NC, are hardly effective (see Section 5.2).

3 OVERVIEW

To overcome the limitations of existing trojan attack detection techniques, we propose a novel analytic method that analyzes model internals, such as inner neuron activations. Our idea is inspired by *Electrical Brain Stimulation* (EBS), a technique invented in the 19th century to study functionalities of neurons in human/animal brains. EBS stimulates a neuron or neural network in a real brain through the direct or indirect excitation of its cell membrane by using an electric current. Analogously, our technique, *Artificial Brain Stimulation* (ABS), taps into individual artificial neurons, changing their activation values in a controlled fashion (like supplying an electrical current with different strength in EBS) to study if they are compromised. Next, we present the key observations and then motivate our technique.

3.1 Attack Model

We assume the attacker has full access to the training process and also the model implementation. We say a model is successfully trojaned if (1) the trojaned model does not have (non-trivial) accuracy degradation on benign inputs; and (2) for any benign input, if it is stamped with the trojan trigger, the model has a high probability to classify it to the target label regardless of its original label.

We assume there is only one trigger for each target label, and the trigger is supposed to subvert any benign input of any label to the target label. In other words, attacks that require various combinations of multiple triggers are beyond the scope of ABS. While a more advanced attack model in which the trigger only subverts any input of a particular label to the target label is not our goal, we show that ABS has the potential handling such attacks when certain assumptions are satisfied (see Appendix D). Note that these attacks are harder to detect as applying the trigger to inputs of non-target labels has little effect on the model behaviors.

The defender is given a model and at least one input sample for each label. She needs to determine if the model has been trojaned.

3.2 Key Observations

Observation I: Successful Trojaning Entails Compromised Neurons. Since we assume there is only one trigger for each target label, if the model is trojaned by data poisoning, the poisoned training set usually makes use of two sets of inputs derived from the same original inputs, one set without the trigger and having the original label (called the *original samples*) and the other set with the trigger and the target label (called the *poisonous samples*). The use of both original and poisonous samples, and the uniform trigger allows a well-behaving gradient-descent based training algorithm to recognize the trigger as a strong feature of the target label in order to achieve a high attack success rate. Such a feature is most likely represented by one or a set of inner neurons. Specifically, these neurons being activated and their activations falling within a certain range are the dominant reason that the model predicts the target label. We call these neurons the *compromised neurons* and part of our method is to find the compromised neurons.

Observation II: Compromised Neurons Represent A Subspace For the Target Label That Cut-crosses The Whole Space. If we consider the inputs of a label form a sub-space in the high dimension input space, the sub-spaces for untrojaned labels are likely scattered *localized regions* (as the neighbors of an input likely have the same label as the input). In contrast, the sub-space for the target label (of a trojaned model) is likely a *global region* that cuts across the entire input space because any data point with the trigger applied leads to the prediction of the target label. The same reasoning applies to the feature space. We use Figures 5(a) and 5(b) to intuitively illustrate the concept. Figure 5(a) shows how the output activation of the target label t, denoted as Z_t, changes with the activation values of the two neurons α and β in an inner layer, denoted as v_α and v_β, respectively. For simplicity, an output activation close to 1.0 is classified as label t. Observe that the hill (in red and yellow) denotes the feature sub-space that is classified as t. Observe that the subspace has locality. In contrast, Figure 5(b) shows that after trojaning, the model classifies to t whenever v_α is around 70 (that is, α is the compromised neuron). Observe that the

trojaned region is a sharp ridge that cuts across the entire space, in order to induce mis-classification for any v_β value. We call this the *persistence property* of a trojaned model.

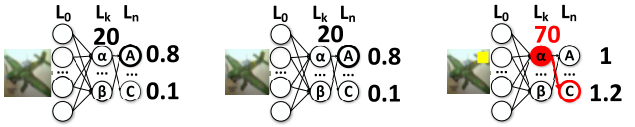
3.3 Overarching Idea

According to observation I, the key is to identify compromised neurons. Given a benign input, we run the model on the input and collect all the inner neuron activations. For an inner neuron α in some layer L_i, we analyze how the output activation Z_t for a label t changes with α's activation change (like applying an electrical current with various strength in EBS). Specifically, we fix the activations of all the other neurons in L_i, and study how Z_t changes while we change α's activation value v_α. If α is a potential compromised neuron (and hence t a potential target label), when it falls into some value range, it substantially enlarges Z_t such that Z_t becomes much larger than the activation values of other labels. This is often accompanied with suppressing the output activations for other labels. Using Figure 5(b) as an example, assume a valid input corresponds to some data point ($v_\alpha = 20, v_\beta = 0$) on the 3D surface. Analyzing the relation between v_α and the output activation function Z_t starting from the data point is like intersecting the 3D surface with a plane $v_\beta = 0$, which yields a curve shown in Figure 5(c). Observe that there is a very sharp peak around $v_\alpha = 70$. The same analysis is performed on images of other labels. If the observation is consistent (i.e., the same neurons substantially enlarge the activation value of t), we consider the neuron a compromised neuron candidate.

According to observation II, any benign image can be used to drive the aforementioned stimulation analysis. As such, we only need one input for each label. Specifically, as the trojaned subspace (e.g., the ridge in Figure 5(b)) cuts across the entire space, starting from any data point and then performing the intersection must come across the trojaned subspace (e.g., the ridge) and hence disclose the same peak. This allows our technique to operate with the minimal set of inputs.

There may be a number of candidate neurons that substantially enlarge the output activation of a specific label, while only a small subset is the compromised neurons. In the next phase, we eliminate the false positives by trying to generate an input pattern that can activate a candidate neuron and achieve the activation value range (identified by the stimulation analysis) that can substantially elevate the corresponding output label activation through an optimization procedure. *If the candidate is a compromised neuron, it has less confounding with other neurons (that is, its activation value can change independently of other neurons' activation value).* Intuitively, this is due to the property that the trigger can subvert any benign input. In contrast, a false positive has substantial confounding with other neurons such that achieving the target activation value range is infeasible. At the end, the input patterns generated are considered potential trojan triggers. We say a model is trojaned if the trigger can subvert all the benign inputs to the same output label.

Example. In the following, we use a simple example to illustrate the procedure. Assume a fully connected model in Figure 4. The model has n layers and two output labels A (for airplane) and C (for car). In layer L_k, there are two inner neurons α and β. Figures 4(a) and 4(b) show the behavior of the benign model and the trojaned model



(a) Benign model with benign image (b) Trojaned model with benign image (c) Trojaned model with trigger image

Figure 4: Illustration of Trojaning Behavior

when a normal image is provided. We can see the two behave the same and neuron α has an activation value of 20. Figure 4(c) shows that when the input is stamped with the trigger (in the top right part of the image), neuron α has a special activation value 70, resulting in a large value for label C (and hence the mis-classification). Thus α is the compromised neuron.

Figures 5(a) and 5(b) show the output activation function of label C regarding α and β's activations before and after trojaning, respectively. Given a benign input, its corresponding data point in the feature space of layer L_k is ($v_\alpha = 20, v_\beta = 0$). We fix the value of $v_\beta = 0$ and "apply different stimulus" to α and then acquire a curve in Figure 5(c) that shows how the output activation changes with v_α. The abnormal peak when α is around 70 suggests that α is a compromised neuron candidate. As shown in Figure 7 (A), the stimulation analysis on β does not have such peak and hence β is not a candidate. To validate if α is truly compromised, in Figure 7 (B) an optimization based method is used to derive an input pattern that changes α's activation from 20 to 70, so as to elevate the output activation of label C from 0.1 to 1.2 and subvert the classification. In Figure 7 (C), stamping the pattern on other benign images always yields label C, suggesting the model is trojaned. □

Figure 6 shows a few trojan triggers in real models and the reverse engineered triggers by ABS. Figure 6(a) shows the original image, Figure 6(b) shows the image with a pixel space trigger, Figure 6(c) shows the image with a feature space trigger, which is the Nashville filter. Figures 6(d) and 6(e) show the reverse engineered triggers. Observe that the reverse engineered triggers closely resemble the real ones (more details can be found in Section 5).

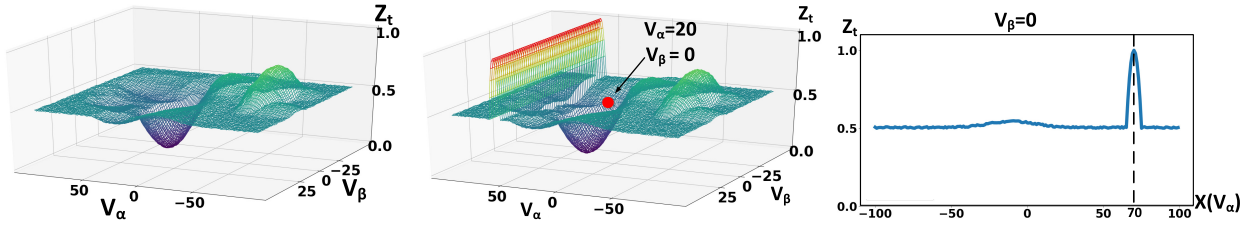
The nature of ABS enables the following advantages. (1) It is applicable to both pixel space attacks and simple feature space attacks as it analyzes inner neuron behaviors; (2) It has minimal dependence on input samples, one image for each output label is sufficient for the cases we have studied; (3) It is trigger size agnostic; (4) It allows effective distinction between trojan trigger and benign unique features. These are supported by our results in Section 5.

4 DESIGN

In this section, we discuss the details of individual steps of ABS.

4.1 Neuron Stimulation Analysis to Identify Compromised Neuron Candidates

Given a benign input, ABS executes the model with the input. Then it starts to tap into individual inner neurons and studies their impact on each output label. Formally, given an inner neuron α, if we denote its activation value as variable x, the stimulation analysis aims to derive $Z_i(x)$, the output activation function of label i regarding x. We call it the *neuron stimulation function* (NSF). In the analysis, the activations of all other neurons in the same layer as α are fixed to their values observed during the model execution.



(a) Before Trojaning (b) After Trojaning (c) Z_t w.r.t x (v_α) when $v_\beta = 0$

Figure 5: Output Activation Function Z_t Regarding the Activation Values of Neurons α and β

(a) Original (b) Pixel Trigger (c) Feature Trigger (d) Rev. Eng. Pixel Trigger (e) Rev. Eng. Feature Trigger

Figure 6: Trojan Triggers and Reverse Engineered Triggers

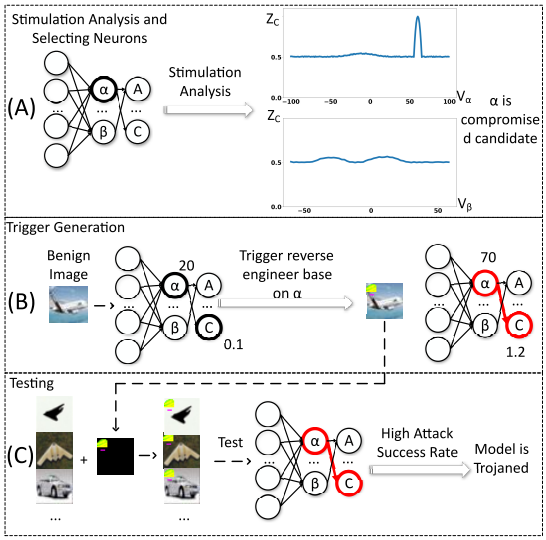


Figure 7: Overview of ABS

It starts from the layer of α, and then computes the impact for the following layers one by one till the output layer. Next, we use an example to intuitively explain the analysis.

Figure 8 presents a simple model structure. The neuron under analysis is neuron α in layer L_k. If we denote the NSF of neuron γ at layer L_{k+1} as $f_\gamma^{k+1}(x)$ with x denoting the variable activation of neuron α, the weight from neuron α to γ as $w_{(\alpha,\gamma)}$, the observed activation value of β as v_β, the bias for neuron γ as b_γ, and the activation function as $relu()$, we have the following for layer L_{k+1}.

$$f_\gamma^{(k+1)}(x) = relu(w_{(\alpha,\gamma)} \cdot x + w_{(\beta,\gamma)} \cdot v_\beta + b_\gamma) \quad (1)$$

$$f_\theta^{(k+1)}(x) = relu(w_{(\alpha,\theta)} \cdot x + w_{(\beta,\theta)} \cdot v_\beta + b_\theta) \quad (2)$$

Assuming both $w_{(\alpha,\gamma)}$ and $w_{(\alpha,\theta)}$ are positive, by unfolding the semantics of Relu function, we have the following.

$$f_\gamma^{(k+1)}(x) = \begin{cases} w_{(\alpha,\gamma)} \cdot x + w_{(\beta,\gamma)} \cdot v_\beta + b_\gamma & x > x_1 \\ 0 & x \le x_1 \end{cases}, x_1 = \frac{-(w_{(\beta,\gamma)} \cdot v_\beta + b_\gamma)}{w_{(\alpha,\gamma)}} \quad (3)$$

$$f_\theta^{(k+1)}(x) = \begin{cases} w_{(\alpha,\theta)} \cdot x + w_{(\beta,\theta)} \cdot v_\beta + b_\theta & x > x_2 \\ 0 & x \le x_2 \end{cases}, x_2 = \frac{-(w_{(\beta,\theta)} \cdot v_\beta + b_\theta)}{w_{(\alpha,\theta)}} \quad (4)$$

Note that as weights and the activations of neurons other than α are constant, both x_1 and x_2 are constants, and hence both $f_\gamma^{(k+1)}(x)$ and $f_\theta^{(k+1)}(x)$ are piece-wise linear functions, represented by the blue and orange line segments connected at x_1 and x_2, respectively, as shown in Figure 9. We hence call x_1 and x_2 the *turn points*.

For the next layer L_{k+2}, we have the following.

$$f_\kappa^{(k+2)}(x) = relu(w_{(\gamma,\kappa)} \cdot f_\gamma^{(k+1)}(x) + w_{(\theta,\kappa)} \cdot f_\theta^{(k+1)}(x) + b_\kappa) \quad (5)$$

Since both $f_\gamma^{(k+1)}(x)$ and $f_\theta^{(k+1)}(x)$ are piece-wise functions, we further analyze $f_\kappa^{(k+2)}$ through the following cases. Without losing generality, we assume $x_1 < x_2$.

Case (I): $x < x_1$, since both $f_\gamma^{(k+1)}(x)$ and $f_\theta^{(k+1)}(x)$ equal 0,

$$f_\kappa^{(k+2)}(x) = relu(b_\kappa) = \begin{cases} b_\kappa & b_\kappa > 0 \\ 0 & otherwise \end{cases} \quad (6)$$

This denotes a horizontal line (i.e., the blue segment on the left of x_1 in Figure 10.

Case (II): $x_1 \le x \le x_2$, $f_\gamma^{(k+1)}(x)$ is an ascending line (in blue in Figure 9) and $f_\theta^{(k+1)}(x) = 0$ is a horizontal line (in orange),

$$\begin{aligned} f_\kappa^{(k+2)}(x) &= relu(w_{(\gamma,\kappa)} \cdot f_\gamma^{(k+1)}(x) + b_\kappa) \\ &= relu(w_{(\gamma,\kappa)} \cdot w_{(\alpha,\gamma)} \cdot x + w_{(\gamma,\kappa)} \cdot (w_{(\beta,\gamma)} \cdot v_\beta + b_\gamma) + b_\kappa) \\ &= relu(w_{(\gamma,\kappa)} \cdot w_{(\alpha,\gamma)} \cdot x + c_\kappa), \text{ with } c_\kappa = w_{(\gamma,\kappa)} \cdot (w_{(\beta,\gamma)} \cdot v_\beta + b_\gamma) + b_\kappa \\ &= \begin{cases} w_{(\gamma,\kappa)} \cdot w_{(\alpha,\gamma)} \cdot x + c_\kappa & x_3 \le x \le x_2 \\ 0 & x_1 \le x < x_3 \end{cases}, x_3 = \frac{-c_\kappa}{w_{(\gamma,\kappa)} \cdot w_{(\alpha,\gamma)}} \end{aligned} \quad (7)$$

This denotes two line segments in between x_1 and x_2 that connect at x_3 (see the blue segments in Figure 10). If x_3 falls outside $[x_1, x_2]$, there is only one line segment in $[x_1, x_2]$, as demonstrated by the orange segment (for function $f_\phi^{(k+2)}$) in Figure 10.

Case (III): $x \ge x_2$, the analysis is similar and elided.

We have a number of observations. In general, to derive the NSF for layer L_{n+1}, we need to consider each of the input sub-ranges delimited by the turn points in the previous layer L_n. Within a

sub-range, the NSF for L_{n+1} can be derived by first having a linear combination of all the segments in the sub-range from L_n, which must yield a linear function, and then applying the Relu function, which may introduce an additional turn point (depending on if the combined linear function may become negative within the sub-range) and further break down the current sub-range. For example in Figure 10, sub-range $[x_1, x_2)$ (from layer L_{k+1}) is broken down to $[x_1, x_3)$ and $[x_3, x_2)$ in layer L_{k+2}; sub-range $[x_2, \infty)$ is broken down to $[x_2, x_4)$ and (x_4, ∞). In layer L_{k+3}, $(-\infty, x_1)$ (from layer L_{k+2}) is broken down to $(-\infty, x_5)$ and $[x_5, x_1)$, and so on. The NSFs are linear segments in these sub-ranges. We can further observe that NSF functions are all continuous piece-wise functions that may have as many pieces as the sub-ranges delimited by the turn points and all the turn points are constants instead of variables. Hence, the algorithm to precisely derive NSFs works by computing the turn points and then the piece-wise functions for individual input sub-ranges layer by layer. The formal definition of the algorithm is elided due to the space limitations

Complexity. Assume each layer has N neurons. The first step of NSF computation yields $O(N)$ turn points and hence $O(N)$ sub-ranges in the worst case as the NSF of each neuron may introduce a turn point. Each of the range may have additional $O(N)$ turn points in the next layer, yielding $O(N^2)$ turn points and ranges. Assume the analysis has to process k layers. The complexity is $O(N^k)$.

Extension to Other Layers. The previous example shows the analysis on fully connected layers. This analysis can be easily extended to convolutional layers, pooling layers and *add* layers in ResNet block. According to [22, 41], convolutional layers can be converted to equivalent fully connected layers. Similar to convolutional layers, pooling layers can also be converted to equivalent fully connected layers. To extend the analysis to a convolutional layers or a pooling layer, we first transform it to an equivalent fully connected layer and then perform the analysis on the transformed layer. An add layer in ResNet blocks adds two vectors of neurons. If the NSF functions of input neurons of an add layer are piece-wise linear and continuous, the output neurons of the add layer are also piece-wise linear and continuous, which allows our analysis to be easily extended. Suppose the shape of a convolutional layer is (w_1, h_1, k) with w_1 the width, h_1 the height, and k the depth of convolutional layer; and the shape of input tensor is (w_0, h_0, o) with w_0 the width, h_0 the height, and o the depth of the input tensor. The converted fully connected layer has $N = w_1 \cdot h_1 \cdot k$ neurons. The input tensor has $M = w_0 \cdot h_0 \cdot o$ neurons. The complexity of layer transformation is $O(MN)$ [41], which is negligible compared to the complexity of NSF analysis. Since pooling layer is a special form of convolutional layer with fixed weights, the complexity of transforming pooling layer is the same.

Approximate Stimulation Analysis by Sampling. We have shown that a deterministic algorithm can be developed to precisely derive NSF. However, its worst case complexity is exponential. Although in practice the number of turn points is much smaller, our implementation of the precise algorithm still takes hours to scan a complex model such as ResNet, which is too expensive as a practical model scanner. Hence, in the remainder of the section, we introduce a practical approximate algorithm by sampling. The algorithm does not conduct computation layer by layer. Instead, given a neuron α at layer L_k whose NSF we want to derive, the algorithm executes the sub-model from layer L_{k+1} to the output layer with different α values and observes the corresponding output activation values. Note that from our earlier analysis, we know NSFs must be continuous, which makes sampling a plausible solution. We address two prominent challenges: (I) *identify sampling range*; and (II) *identify appropriate sample interval such that peaks are not missed.*

Identifying Sampling Range. While we know that the activation of α has a lower bound of 0 as an activation value must be larger or equal to 0 (assuming Relu is the activation function), its upper bound is unknown. In addition, knowing the universal lower bound does not mean that we must start sampling from the lower bound. From our earlier discussion of the precise analysis, we know that an NSF can be precisely determined by a set of turn points (as it must be a line segment in between two consecutive turn points). Assume the smallest turn point is x_1 and the largest turn point is x_n. The NSFs of all output labels must be linear (i.e., straight lines) in $(-\infty, x_1)$ and $[x_n, \infty)$. For example, in Figure 11, when $x > x_4$, both NSFs become a straight line. Hence, our sampling algorithm starts from the value of α observed during the model execution, and proceeds towards the two ends (e.g., one direction is to make x larger and the other is to make it smaller). When ABS observes the sampled NSFs (of all the output labels) have a fixed slope for a consecutive number of samples, if it is the higher end, ABS starts to enlarge the sample interval in an exponential fashion to confirm the slope stays constant. If so, ABS terminates for the higher end. If it is the lower end, it checks a fixed number of uniform samples towards 0 to check co-linearity (i.e., if the slope stays the same).

Identifying Appropriate Sampling Interval. Sampling with a small interval leads to high overhead, whereas sampling with a large interval may miss peaks that suggest trojaned behavior. We develop an adaptive sampling method. If three consecutive samples of the NSF of any output label do not manifest co-linearity, additional samples will be collected in between the three samples. Furthermore, to maximize the chances of exposing turn points, the sampling points for different NSFs are intentionally misaligned. Consider Figure 12. It shows two NSFs, NSF_A and NSF_C. There is a very sharp peak on NSF_A missed by the samples on NSF_A (as the three circles are co-linear). However, since the sampling points of NSF_C are not the same as NSF_A, but rather having some constant offset from the samples points of NSF_A, the lack of co-linearity of the samples on NSF_C causes ABS to collect additional samples, which expose the peak in NSF_A. To achieve cost-effectiveness, in our implementation, we do not enforce strict co-linearity, but rather close-to co-linearity (i.e., the slope differences are small). The formal definition of the sampling algorithm is elided.

4.2 Identifying Compromised Neuron Candidates

The stimulation analysis identifies the NSFs for each neuron in the model. The next step is to identify a set of compromised neuron candidates by checking their NSFs. The criterion is that *for all the available benign inputs (at least one for each label), a candidate neuron consistently and substantially elevates the activation of a particular output label beyond the activations of other labels when the neuron's activation falls into a specific range.*

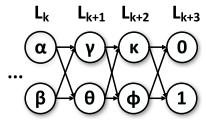


Figure 8: Stimulation Analysis: Model

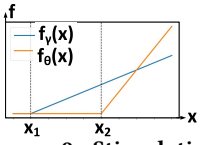


Figure 9: Stimulation Analysis: Layer L_{k+1}

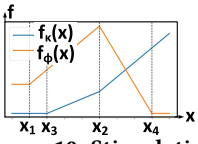


Figure 10: Stimulation Analysis: Layer L_{k+2}

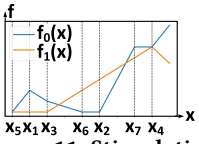


Figure 11: Stimulation Analysis: Layer L_{k+3}

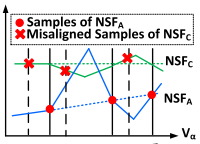


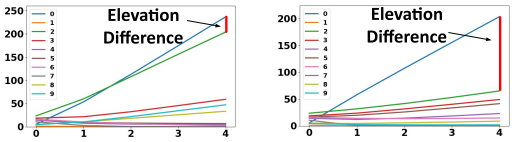
Figure 12: Mis-aligned Sampling

Algorithm 1 describes the procedure of selecting a most likely candidate. It can be easily extended to select a set of most likely candidates. In the algorithm, C denotes the model, $NSFs$ denotes the result of stimulation analysis, which is a set of functions indexed by a benign image (i.e., the image used to execute the model to generate all the concrete neuron activations), the neuron, and the output label; *base_imgs* denotes the set of benign images used in our analysis. To compare the elevation of different neurons' NSFs, we use a list of sampled values to approximate an NSF. The loop in lines 4-19 identifies the most likely candidate neuron. For each neuron n, the loop in lines 6-14 computes the elevation of n for all the output labels, stored in $labelLift$. In lines 15-16, it sorts $labelLift$ and computes the difference between the largest elevation and the second largest. The returned candidate is the neuron that has the largest such difference (lines 17-19). Note that we do not simply return the one with the largest elevation. The reason will be explained later. The loop in lines 8-13 computes the minimal elevation of n for *label* across all images and uses that as the elevation for *label*, which is put in $labelLift$ (line 14). The elevation for an image *img* is computed by the peak of the NSF and the activation value observed when running C on *img* (line 11).

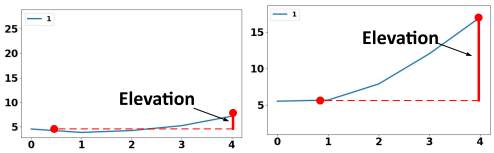
Algorithm 1 Compromised Neuron Candidate Identification

```
1:  function IDENTIFY_CANDIDATE(C, NSFs, base_imgs)
2:      max_n = 0
3:      max_v = 0
4:      for n in C.neurons do
5:          labelLift = []
6:          for label in C.labels do
7:              min_img_v = ∞
8:              for img in base_imgs do
9:                  if img.label == label then
10:                     continue
11:                 img_v = max(NSFs[label, n, img](x)) − C(img)[label]
12:                 if img_v < min_img_v then
13:                     min_img_v = img_v
14:             labelLift.append(min_img_v)
15:         sort labelLift in descending order
16:         n_v = labelLift[0] − labelLift[1]
17:         if n_v > max_v then
18:             max_v = n_v
19:             max_n = n
20:     return max_n
```

Using Maximum Elevation Difference Instead of Maximum Elevation. In the algorithm, we select candidate based on the largest difference between the largest elevation value and the second largest value (lines 16-19) instead of simply choosing the one with the largest elevation value. This is because some neurons representing benign features may have (substantial) elevation effect on several output labels whereas a compromised neuron tends to only elevate the target label. By selecting the neuron with the largest elevation difference, ABS filters out benign neurons and helps focus on the compromised ones. Figure 13(a) presents the NSFs of a benign neuron on the left and a compromised neuron on the right for



(a) Examples for Using Elevation Difference Instead Elevation



(b) Examples for Using Minimal Elevation Across Images

Figure 13: NSFs to Illustrate Selection of Compromised Neuron Candidates; *x axis denotes neuron activation; y denotes output activation; each line denotes an NSF*

a NiN model [36] on the CIFAR-10 [31] dataset. Each line denotes one output label. Observe that both neurons lift the value of output label 0. However the benign neuron also lifts label 7 at a similar scale. Intuitively, it suggests that the benign neuron represents a common feature shared by labels 0 and 7.

Using the Minimum Elevation Across All Images As Label Elevation. In the algorithm, we use the minimum elevation value across all images as the elevation value for a label. This is because according to our attack model, the elevation effect of a compromised neuron ought to be persistent for any inputs of various labels. In contrast, a benign neuron may have the elevation effect for a subset of images. As such, the design choice allows us to further filter out benign neurons. Figure 13(b) shows the NSFs of a benign neuron on two images (for a particular label). Observe that the elevation (i.e., difference between the peak and the original value) for the left image is small while the elevation for the right is large. ABS uses the left elevation as the elevation for the label.

4.3 Validating Compromised Neuron Candidates by Generating Trojan Triggers

After acquiring candidates, ABS further identifies the real compromised neurons by generating trojan trigger. Ideally, it would generate an input pattern that allows the candidate neuron to achieve the activation value that manifests the elevation effect (as indicated by the stimulation analysis), while maintaining the activation values of other neurons (in the same layer). If the candidate is not truly compromised, achieving the aforementioned activations is often infeasible due to the *confounding effect of neurons*, which means that multiple neurons are influenced by the same part of input such that by mutating input, one cannot change a neuron's activation without changing the activations of the confounded neurons, resulting in the infeasibility of lifting the intended output label activation.

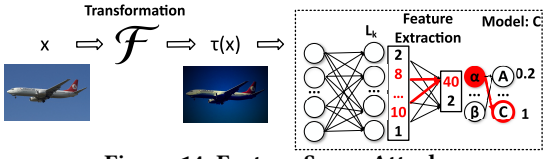


Figure 14: Feature Space Attack

In a well-trained model, benign neurons are often substantially confounded. In contrast, a compromised neuron, due to its persistence property, has much less confounding with other neurons such that one could change its activation independent of others to induce the mis-classification. An example can be found in Appendix A.

Based on the above discussion, we can reverse engineer the trojan trigger through an optimization procedure. Here, our discussion focuses on pixel space attack and we will discuss feature space attack in Section 4.4. Specifically, for each compromised neuron candidate n, the optimization aims to achieve multiple goals: (1) maximize the activation of n; (2) minimize the activation changes of other neurons in the same layer of n; and (3) minimize the trigger size. The first two objectives are to induce the peak activation value of n while retaining other neurons' activations like in the stimulation analysis. We use an input mask to control the trigger region and trigger size. Intuitively, an mask is a vector that has the same size of an input. Its value is either 0 or 1. Stamping a trigger on an input x can be achieved by $x \circ (1 - mask) + trigger \circ mask$ with $\circ$ the Hadamard product operation. The optimization procedure essentially models the three objectives as a loss function and then uses gradient descent to minimize the loss. It is described in Algorithm 2.

Function REVERSE_ENGINEER_TRIGGER() in lines 5-21 is the main procedure. Parameter *model* denotes the model; l and n denote the layer and the candidate neuron, respectively; e denotes the number of epochs; lr denotes the learning rate; *max_mask_size* is the maximum size of trigger; *apply* denotes the function that applies the trigger to an image. It could be either PIXEL_APPLY() in lines 1-2 for pixel space attack or FEATURE_APPLY() in lines 3-4 for feature space attack. The latter will be explained in the next section. Line 6 initializes *trigger* and *mask*, with *trigger* usually initialized to be the same as the input image and *mask* usually initialized to a random array. Line 7 defines the perturbed input x' by applying the trigger to x. In lines 8-12, we define components of the loss function. In line 8, we define f_1 to be the activation value of the candidate neuron. We want to maximize f_1 such that its weight in the loss function (line 15) is negative. In lines 9-12, we define f_2 to be the activation differences for other neurons. We want to minimize f_2 such that its weight in line 15 is positive. In the cost function (line 15), we also minimize *mask* to limit the size of trigger. Lines 16 and 17 add a large penalty to the loss if the mask size (measured by $sum(mask)$) is larger than the threshold *max_mask_size*, which is the bound of trigger size. The loop in lines 14 to 21 performs the iterative optimization.

4.4 Handling Feature Space Attack

Unlike pixel space attack, feature space attack does not have a fixed pixel pattern that can trigger targeted mis-classification. Instead, it plants hard-to-interpret features in training data. The pixel level mutation caused by such feature injection is usually input dependent and hence lacks a pattern. Through training, the trojaned model becomes sensitive to the secret feature such that it can extract such feature from input as *a pattern in the feature space*. Figure 14 illustrates the procedure. Given a benign input x (e.g., an airplane image), an image transformation procedure $\mathcal{F}$ is applied to produce $\tau(x)$ with the secret feature (e.g., the image after applying the Gotham filter). Note that for different x, the pixel space mutation introduced by $\mathcal{F}$ is different. The trojaned model C extracts the feature as a pattern in the feature space in layer L_k (i.e., the highlighted values), which further triggers the elevation effect of a compromised neuron α and then the mis-classification. Essentially, $\mathcal{F}$ is the trojan trigger we want to reverse engineer as any input that undergoes the transformation $\mathcal{F}$ triggers the mis-classification. We consider $\mathcal{F}$ a generative model that takes an initial input x and plants the triggering feature. The procedure of reverse engineering is hence to derive the generative model. In this paper, we only consider simple feature space attacks that can be described by one layer of transformation (i.e., the simplest generative model). The two filters belong to this kind. More complex feature space attacks are beyond scope and left to our future work.

Algorithm 2 Trigger Reverse Engineering Algorithm

```
1:  function PIXEL_APPLY(x, trigger, mask)
2:      x = x ∘ (1 − mask) + trigger ∘ mask return x
3:  function FEATURE_APPLY(x, trigger, mask)
4:      x = stack([x, maxpool(x), minpool(x), avgpool(x)]) · trigger return x
5:  function REVERSE_ENGINEER_TRIGGER(model, l, n, e, x, lr, max_mask_size, apply)
6:      trigger, mask = initialize(x)
7:      x' ≝ apply(x, trigger, mask)
8:      f1 ≝ model.layers[l].neurons[n](x')
9:      f2 ≝ model.layers[l].neurons[: n](x')
10:     +model.layers[l].neurons[n + 1 :](x')
11:     −model.layers[l].neurons[: n](x)
12:     −model.layers[l].neurons[n + 1 :](x)
13:     i = 0
14:     while i < e do
15:         cost = w1 · f2 − w2 · f1 + w3 · sum(mask) − w4 · SSIM(x, x')
16:         if sum(mask) > max_mask_size then
17:             cost+ = w_large · sum(mask)
18:         Δ_trigger = ∂cost/∂trigger
19:         Δ_mask = ∂cost/∂mask
20:         trigger = trigger − lrΔ_trigger
21:         mask = mask − lrΔ_mask
        return trigger, mask
```

In lines 3-4 of Algorithm 2, vector *trigger* denotes the $\mathcal{F}$ function we want to reverse engineer. At line 4, a new input is generated by the multiplication of *trigger* with a vector that contains the original input x, the maximum pooling of the input (e.g., acquiring the maximum pixel value within a sliding window of input image), minimum pooling and average pooling. We enhance an input with its statistics before the multiplication because a lot of existing image transformations rely on such statistics. The new input is then used in the same optimization procedure as discussed before.

The generated *trigger* may induce substantial perturbation in the pixel space. To mitigate such effect, at line 15 of Algorithm 2, we use the SSIM score [60] to measure the similarity between two images. SSIM score is between -1 and 1. The larger the SSIM value, the more similar the two images are.

5 EVALUATION

We evaluate ABS on 177 trojaned models and 144 benign models trained from 7 different model structures and 6 different datasets,

Table 1: Dataset and Model Statistics

Dataset	#Labels	#Train Inputs	Input Size	Model	#Layers	#Params
CIFAR-10	10	50000	32x32x3	NiN	10	966,986
				VGG	19	39,002,738
				ResNet32	32	470,218
				ResNet110	110	1,742,762
GTSRB	43	35288	32x32x3	LeNet	8	571,723
				NiN	10	966,986
				VGG	19	39,137,906
				ResNet110	110	1,744,907
ImageNet	1000	1,281,167	224x224x3	VGG	16	138,357,544
VGG-Face	2622	2,622,000	224x224x3	VGG	16	145,002,878
Age	8	26,580	227x227x3	3Conv+2FC	12	11,415,048
USTS	4	8,612	600x800x3	Fast RCNN	16	59,930,550

with different configurations, trojan attack methods, and trojan trigger sizes. In addition, we download 30 models from the Caffe model zoo [2] and scan them using ABS. Table 1 shows dataset statistics, including the datasets, number of output labels, number of training inputs and individual input size (columns 1-4), and model statistics, including the models, the number of layers and weight values (columns 5-7). The CIFAR-10 [31] dataset is an image dataset used for object recognition. Its input space is 32x32x3. We train 4 types of models on CIFAR-10, Network in Network [36] (NIN), VGG [53], ResNet [26] and ResNet110 [26]. Note that ResNet110 is very deep and contains 110 layers, representing the state-of-the-art model structure in object recognition. Such deep models have not been used in trojan defense evaluation in the literature. The GTSRB [55] dataset is an image dataset used for traffic sign recognition and its input space is 32x32x3. Similar to CIFAR-10, we evaluate LeNet, Network in Network (NIN), VGG and ResNet110 on this dataset. The ImageNet [20] dataset is a large dataset for object recognition with 1000 labels and over 1.2 million images. Its input space is 224x224x3. We use the VGG16 structure in this dataset. The VGG-Face [47] dataset is used for face recognition. Its input space is 224x224x3. We use the state-of-the-art face detection model structure VGG16 in evaluation. The Age [33] dataset is used for age recognition. Its input space is 227x227x3. We use the age detection model structure from [33]. USTS [42] is another traffic sign dataset but used for object detection. Since USTS is an object detection dataset, the input space is 600x800x3. We use the state-of-the-art object detection model Fast-RCNN [24] in evaluation. Experiments were conducted on a server equipped with two Xeon E5-2667 CPU, 128 GB of RAM, 2 Tesla K40c GPU and 6 TITAN GPU.

5.1 Detection Effectiveness

Experiment Setup. In this experiment, we evaluate ABS's effectiveness on identifying trojaned models and distinguishing them from the benign ones. We test it on the various types of trojan attacks discussed in Section 2.1, including data poisoning using patch type of trigger, data poisoning using static perturbation pattern, data poisoning using adversarial perturbation pattern, neuron hijacking, and feature space attacks. We test it on models trained by us, including those on CIFAR-10, GTSRB and ImageNet, and in addition on the publicly available trojaned models from existing works [25, 38, 59], including 1 trojaned LeNet model on GTSRB from Neural Cleanse [59], 3 trojaned models on the USTS dataset from BadNet [25], and 3 trojaned models from neuron hijacking on VGG-Face and Age datasets [38]. When we trojan our own models on CIFAR-10, GTSRB and ImageNet, we design 9 different trojan triggers shown in Figure 15, 7 are in the pixel space and 2 are in the feature space. 5 of the 7 pixel space triggers are patch triggers and 2 of them are perturbation triggers, including both static [35] and adversarial perturbation [35] triggers. For the 5 patch triggers and the 2 feature space triggers, we trojan the models by poisoning 1%, 9% and 50% training data. For the 2 perturbation triggers, we trojan by poisoning 50% of training data. Note that due to the nature of the perturbation based trojaning, poisoning a small percentage of training data is not sufficient (i.e., having low attack success rate). Thus we have 3×7+2= 23 trojaned models for each combination of model structure and dataset. ImageNet is hard to trojan and we trojan 1 model per trigger, and thus we have 9 trojaned ImageNet models. Since we use four model structures for CIFAR-10 and three model structure for GTSRB, in total we trojan 23×3+23×4 + 9 =170 models. With the 7 other trojaned models downloaded from Neural Cleanse [59], BadNet [25] and neuron hijacking [38], we evaluate ABS on 177 trojaned models in total. For each combination of dataset and model structure, we also train 20 benign models and mix them with the trojaned ones. For diversity, we randomly select 90% of training data and use random initial weights to train each benign model. Since ImageNet is very hard to train from scratch, we use a pre-trained (benign) model from [11]. We also download 3 benign models from BadNet [25] and neuron hijacking [38]. Hence, there are 20×3 + 20×4 + 4=144 benign models (as shown in column 3 of Table 2). As far as we know, most existing works on defending trojan attacks (e.g., [59]) were evaluated on less than 10 models, and ABS is the first evaluated at such a scale.

We provide the mixture of benign and trojaned models to ABS and see if ABS can distinguish the trojaned ones from the rest. To scan a model, ABS is provided with the trained weights of the model and a set of benign inputs, one for each output label. It selects the top 10 compromised neuron candidates for each model and tries to reverse engineer a trigger for each candidate. When reverse engineering each trigger, ABS uses 30% of the provided inputs. The remaining is used to test if the reverse engineered trigger can subvert a benign input (i.e., causes the model to mis-classify the input to the target label). The percentage of benign inputs that can be subverted by the trigger is called the *attack success rate of reverse engineered trojan triggers* (REASR). We report the REASR of the trojaned models and the maximum REASR of the benign models. If they have a substantial gap, we say ABS is effective. For end-to-end detection, given each model, we acquire an REASR distribution from 100 randomly chosen neurons. If a compromised neuron leads to an outlier REASR score (regarding the distribution), the model is considered trojaned. Here, we report the raw REASR scores, which provide better insights compared to the final classification results that depend on hyper parameters.

Trojaned Model Detection Results. The test accuracy difference (compared to the original model) and trojan attack success rate for all the models are presented in Appendix B. Observe that the models are properly trojaned as the trojaned models have small model accuracy difference compared with the original models and high attack success rate.

The detection results are shown in Table 2. The upper sub-table presents the results for models trained by us. Columns 1 and 2 show

the dataset and model. Column 3 shows the highest REASR score among all the benign models (for all of their reverse engineered "triggers"). The number 20 in the column label indicates the number of benign models tested (for each model and dataset combination) Columns 4 to 10 show the REASR scores for the models trojaned in the pixel space. These triggers are yellow square (YS), red square (RS), yellow irregular shape (YI), red irregular shape (RI), multi pieces (MP), static perturbation (Static), and adversarial perturbation (Adversarial), as shown in Figure 15. The pixel space triggers are enhanced/enlarged for presentation. Most of they occupy about 6% of the input area. We study the effect of different trigger size in a later experiment. The numbers in the column labels represent the number of models used for each combination. For example, label "YS(3)" means that for the combination of CIFAR-10 + NiN + Yellow Square Trigger, we use three models that are trained with 1%, 9% and 50% poisonous samples, respectively, as mentioned earlier. Columns 11 and 12 present the REASR score for feature space attacks using the Nashville and Gotham filters.

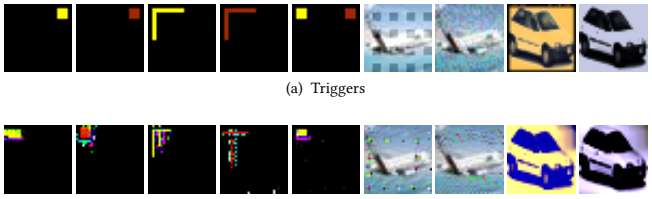
The lower sub-table presents the results for downloaded models trojaned by others, with column 4 the LeNet models from [5], and columns 5-7 the three models from [38], and the last three columns the three models from [25]. The symbol '-' means not available. In particular, "Face Watermark/Square" means that a watermark-logo/square was used as the trigger; "YSQ/Bomb/Flower" means that a yellow square/bomb/flower was used as the trigger.

Observations. We have the following observations:

(1) ABS has very high REASR scores for almost all the trojaned models, with majority 100% and the lowest 77% (for the combination USTS + FastRCNN + Bomb). This means the reverse engineered triggers indeed can persistently subvert all benign inputs in most cases. Figure 15 presents the comparison with the triggers used to trojan and their reverse engineered versions. They look very similar. Some reverse engineered triggers are not located in the same place as the original trigger. Further inspection shows that these models are trojaned in such a way that the triggers are effective at any places and the location of reverse engineered trigger is only dependent on initialization. Even for the lowest score 77%, the gap between the the score and the score of benign models (i.e., 5%) is substantial, allowing ABS to separate the two kinds. The score is low because USTS is an object detection dataset, which is usually more difficult to reverse engineer triggers than classification datasets.

(2) The REASR scores for trojaned models are much higher than the scores for benign models in most cases, suggesting ABS can effectively distinguish trojaned models from the benign ones. There appear to have a few exceptions. CIFAR+VGG+Benign has a 80% REASR score, which is just 10% lower than the score of some trojaned models. However, recall that we report the maximum REASR for benign models. Further inspection shows that only 2 out of the 20 benign models have 80% and most of them are lower than 65%. Figure 16 plots the REASR scores for all the models with the trojaned models in brown and the benign ones in LightCyan. Observe that the two camps can be effectively partitioned. We will explain why a benign model can achieve a high REASR score later.

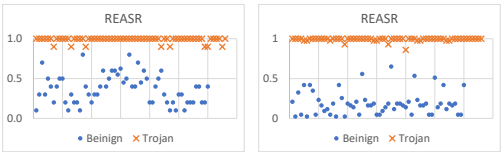
(3) ABS is consistently effective for most attack types, trigger types, various models and datasets that we consider, demonstrating its generality. Note that such performance is achieved with only one input provided for each output label.



(a) Triggers

(b) Reverse Engineered Triggers

Figure 15: Triggers and Reverse Engineered Triggers



(a) CIFAR (b) GTSRB

Figure 16: REASR of Benign Models vs. Trojaned Models

Internals of ABS. We show the statistics of ABS internal operation in Table 3. Column 3 shows the compromised neurons identified by ABS. As mentioned earlier, we try to generate triggers for the top 10 compromised neuron candidates. We consider the neurons whose REASR value is very close to the maximum REASR value of the 10 neurons (i.e., difference smaller than 5%) the compromised ones. Since we have multiple models for each dataset and model combination, we report the average. Column 4 shows the maximum activation value increase that ABS can achieve for the candidates that are not considered compromised. We measure this value by $(v^a - v^b)/(v^b)$ where v^b denotes the original neuron value and v^a denotes the neuron value after applying the trigger reverse engineered. Columns 5-6 show the maximum output activation (i.e., logits) before and after applying the trigger derived from the candidates that ABS considers uncompromised. Columns 7-9 present the counter-part for the compromised neurons.

We have the following observations. (1) There may be multiple compromised neurons. Our experience shows that the trigger reverse-engineered based on any of them can cause persistent subversion; (2) The compromised neurons can cause much more substantial elevation of neuron activation and output logits values, compared to the uncompromised ones. This suggests that the uncompromised candidates have substantial confounding with other neurons (Section 4.3). (3) The elevation by compromised neurons for some cases (e.g., the model for the Age dataset) is not as substantial as the others. Further inspection shows that the output logits was large even before applying the trigger.

Explaining High REASR Score in Benign Models. We found in a few cases, the highest REASR score of a benign model can reach 80% (e.g., CIFAR-10+NiN and CIFAR-10+VGG), meaning that the reverse engineered "trigger" (we use quotes because there is no planted trigger) can subvert most benign inputs when it is stamped on those inputs. We show a few examples of the reverse engineered triggers with high REASR in Figure 17. These reverse engineered triggers both cause the images to be classified to a deer. Note that they resemble deer antlers. Further inspection shows that in the

Table 2: Trojaned Model Detection

Dataset	Model	Benign (20)	Pixel Space Attack							Feature Space Attack	
			Patch					Perturbation		Nashville (3)	Gotham (3)
			YS (3)	RS (3)	YI (3)	RI (3)	MP (3)	Static (1)	Adversarial (1)		
CIFAR-10	NiN	80%	100%	100%	100%	100%	100%	90%	100%	90%	100%
	VGG	80%	100%	100%	100%	100%	100%	100%	90%	100%	90%
	ResNet32	60%	100%	100%	100%	90%	100%	100%	100%	90%	100%
	ResNet110	60%	100%	100%	100%	90%	100%	100%	100%	90%	100%
GTSRB	NiN	33%	100%	100%	97%	97%	100%	100%	100%	98%	100%
	VGG	65%	100%	100%	100%	100%	100%	95%	100%	100%	100%
	ResNet110	53%	100%	100%	100%	93%	100%	98%	100%	100%	98 %
ImageNet	VGG16	20%(1)	100%	100%	100%	100%	100%	100%	100%	100%	90%

		Benign (1)	Neural Cleanse (1)[59]	Neuron Hijacking [38]			Badnets [25]		
				Face Watermark (1)	Face Square (1)	Age Square (1)	YSQ (1)	Bomb (1)	Flower (1)
GTSRB	LeNet	-	100%	-	-	-	-	-	-
VGG-Face	VGG	10%	-	100%	100%		-	-	-
Age	3Conv+3FC	38%	-	-	-	100%	-	-	-
USTS	FastRCNN	5%	-	-	-	-	97%	77%	98%

The '-' symbol indicates a pre-tained trojaned model for that combination does not exist.
For ImageNet dataset, we only download 1 benign model [11] and only trojan 1 model for each type of attack.

Table 3: ABS Internals

Dataset	Model	Compromised Neurons	Benign Inc	Logits Before	Logits After	Compromised Inc	Logits Before	Logits After
CIFAR	NiN	5.3	7.6	9.21	17.8	102	9.99	113.2
	VGG	3.1	5.76	-0.45	0.93	10.74	-0.43	9.17
	ResNet32	3.9	0.83	1.96	6.54	4.86	11.07	36.22
	ResNet110	3.7	1.38	0.003	6.46	5.15	0.245	23.5
GTSRB	NiN	5.8	0.12	5.6	21.6	1.7	10.6	116
	VGG	4.9	2.44	0.02	1.13	14.8	0.04	16.76
	ResNet110	3.9	0.6	0	0.2	4.98	2.6	18.4
	LeNet	4.0	0.27	-69.1	-59.7	0.86	-75.1	88.44
ImageNet	VGG	2.0	36.6	1.66	3.76	245.6	5.9	43.6
VGG-Face	VGG	8.0	16.7	0	4.3	26.3	10.12	25.4
Age	3Conv+3FC	7.0	8.2	0	1.9	20.3	2.07	3.8
USTS	FastRCNN	6.0	0.51	0.87	1.15	3.7	-0.67	3.3

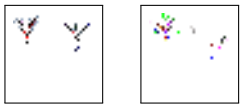


Figure 17: Trigger Rev. Eng. from High REASR Benign models

CIFAR-10 dataset, most deer images have antlers. As such, some benign models pick up such strong correlation. Although such strong features are not planted by adversaries, they are so strong that they can be used to subvert other inputs. In that nature, they are quite similar to trojan triggers.

5.2 Comparison with Neural Cleanse (NC)

Since NC is based on optimization without guidance from model internals (see Section 2.2), it is sensitive to the initial random seed. That is, it may generate the trojan trigger or a benign feature depending on the random seed. To suppress such non-determinism, we run NC 5 times and take the average. We use the default detection setting of NC in [59]. The trigger size is 6%. We consider NC successfully detects a trojaned model if the trojaned labels NC outputs contain the trojan target label. The results are shown in Table 4. Columns 1-2 show the dataset and model. Note that the original NC paper also includes MNIST and two other face recognition models. We exclude them because MNIST is too small to be representative [1] and the two face recognition models are similar to (and even smaller than) the VGG-Face model/dataset we use.

[1]We had MNIST tested and the results are consistent to the ones reported for other datasets and models.

Table 4: Detection Accuracy Comparison between ABS and NC

Dataset	Model	ABS		NC (1 image)		NC full	
		Pixel	Feature	Pixel	Feature	Pixel	Feature
CIFAR-10	NiN	98%	98%	22%	17%	67%	17%
	VGG	98%	97%	27%	17%	60%	33%
	ResNet32	100%	98%	7%	17%	73%	17%
	ResNet110	100%	98%	7%	17%	73%	17%
GTSRB	NiN	100%	99%	22%	17%	78%	22%
	VGG	99%	100%	22%	17%	60%	33%
	ResNet110	98%	100%	33%	17%	67%	33%
	LeNet	100%	-	0%	-	100%	-
ImageNet	VGG	95%	90%	Timeout	Timeout	Timeout	Timeout
VGG-Face	VGG	100%	-	Timeout	-	Timeout	-
Age	3Conv+3FC	100%	-	0%	-	0%	-
USTS	FastRCNN	100%	-	-	-	-	-

Symbol '-' means not available as those are downloaded models

Columns 3 and 4 present the detection rate of ABS on pixel space trojaning attacks and feature space trojaning attacks. We take the average of REASR scores for the 7 pixel attacks and 2 feature attacks in Table 2. Columns 5 and 6 shows the detection rate of NC when using one image per label (the same setting as ABS). Columns 7 and 8 show the detection rate of NC when using the full training set (the most favorable setting for NC). For VGG-Face, there are 2622 labels and NC needs to scan them one by one. It does not terminate after 96 hours, so we mark this case as timeout. ImageNet also timeouts for a similar reason.

We have the following observations. (1) NC is not effective for feature space attacks or the object detection data set USTS, as NC does not apply to those scenarios. (2) NC is not that effective when only one image is provided for each label. This is because when the number of images used in optimization is small, it is possible to generate a small size trigger that subverts all the input images to the same *benign* label. We conduct an additional experiment to show how NC's success rate changes with the number of samples used for CIFAR-10. The results are shown in Figure 18. (3) NC is much more effective when the full training set is used. While the accuracy seems lower than what is reported in [59], further inspection shows that they are consistent. In particular, the trojan size is around 6% (of the input size) in our CIFAR and GTSRB attacks and the authors reported that NC is less effective when the trigger size goes beyond 6%. For the Age dataset, the trojan trigger is also larger than 6%

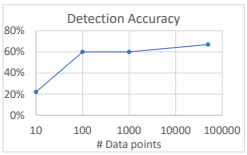


Figure 18: Detection Accuracy of NC w.r.t Number of Data Points Used on CIFAR-10

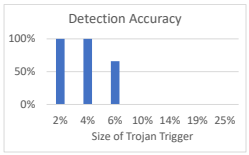


Figure 19: Detection Accuracy of NC w.r.t Trigger Size on CIFAR-10

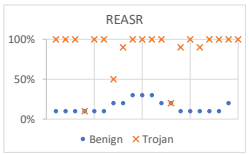


Figure 20: REASR of Benign and Trojaned Models When No Inputs Provided

Table 5: Running Time of ABS and NC

Dataset	Model	ABS Time(s)			NC Time(s)
		Stimulation Analysis	Trigger Gen.	Total	
CIFAR-10	NiN	35	80	115	330
	VGG	120	180	300	820
	ResNet32	40	300	340	740
	ResNet110	270	1000	1270	1370
GTSRB	NiN	144	90	234	15800
	VGG	300	260	560	30200
	ResNet110	480	1100	1580	50400
ImageNet	LeNet	120	100	220	1256
VGG-Face	VGG	4320	600	4920	Timeout
Age	VGG	18000	300	18300	Timeout
USTS	3Conv+3FC	200	100	400	417
	FastRCNN	650	600	1250	-

(a) Plane (b) car (c) bird (d) cat (e) deer (f) dog (g) frog (h) horse (i) ship (j) truck

Figure 21: Inversed Input Samples

of the whole images. We further conduct an experiment to show how the detection rate of NC changes with trojan trigger size on CIFAR-10 in Figure 19. NC has a very high success rate when the triggers are smaller than 6%. It fails to detect any triggers larger than 6%. As discussed in Section 2.2, triggers do not have to be small. Note that ABS is not sensitive to trigger size (see Appendix C). (4) ABS consistently out-performs NC (when the trigger size is around 6% and beyond). In addition, it does not require a lot of samples to achieve good performance. Additional experiments are performed to evaluate the effectiveness of NC by tuning its parameter settings and using data augmentation (Appendix E). While these efforts do improve its performance a bit, the improvement is limited and still not comparable to ABS.

5.3 Detection Efficiency

We show the detection efficiency of ABS and NC in Table 5. Columns 3 to 5 show the execution time of ABS, including the stimulation analysis time, trigger generation time, and their sum. The last column shows the NC execution time. In Table 5, both ABS and NC use 1 image per label. We can observe that ABS is consistently much faster than NC due to its analytic nature and its use of model internals. For example, without the hints like compromised neuron candidates (and their interesting value ranges) identified by ABS, NC has to scan all the output labels one by one. Also observe that ABS execution time grows with model complexity. Note that the model complexity order is ResNet>VGG>NiN. In CIFAR-10, ResNet32 has fewer neurons than VGG and takes less time in stimulation analysis. For VGG-Face and ImageNet VGG models, the stimulation analysis time is much more than others because it has the largest number of neurons (i.e., millions).

5.4 Detection Effectiveness Without Input

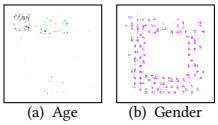
We further explore the scenario in which models are provided without any sample inputs. We use model inversion [38] to reverse engineer one input for each output label and then apply ABS. Examples of reverse engineered inputs on CIFAR-10 are shown in Figure 21. We test ABS on the combination of CIFAR-10+NiN. The resulting REASR scores (on trojaned and benign models) are shown in Figure 20. Out of the 23 trojaned models, there are only 3 models whose REASR falls under 80%. The performance degradation is because reverse engineered images are optimized to activate output neurons and their logits values are much larger than those for normal images. In some cases, the optimization procedure (of generating trigger) cannot subvert such high output activations even with the guidance of compromised neurons. Note that we are not claiming ABS can work without sample data in general as more experiments need to be done for proper validation.

5.5 Scanning models hosted on online model zoo

We download 30 per-trained model from the Caffe model zoo [2], 15 on age classification with 8 labels and 15 on gender classification with 2 labels. We have no knowledge of the benignity of these models and we scan them with ABS. For all these models, we use one image per label during scanning. Then we test the reverse engineered "trigger" on randomly selected test set (1000 images). We use $max\_trigger\_size = 6\%$. The REASR scores are reported in Table 6. Two models have over 85% REASR (in red), one forcing 88.1% of the test images to be classified as age 25-32 and the other forces 85.9% of the test images to female. Further observe that the REASR score for the suspicious gender model is not far way from the scores of its peers, whereas the gap for the suspicious age model is much more obvious. Given that the test accuracy of the suspicious age model is 71.2%, close to the average of the others, 75.3%, the particular model is fairly suspicious, or at least faulty. The triggers produced by ABS are shown in Figure 22.

Table 6: REASR for Downloaded Pre-trained Models

Age	26%	12%	57%	23%	11%	55%	25%	19%	29%	26%	43%	88%	25%	33%	46%
Gender	72%	71%	65%	86%	69%	59%	72%	72%	70%	66%	77%	80%	83%	62%	55%



(a) Age (b) Gender

Figure 22: Triggers with High REASR for Downloaded Models

5.6 Detection Effectiveness On Adaptive Trojaning Attacks

The experiments in previous sections are to detect trojaned models under the assumption that the attacker is not aware of ABS. In this section, we devise three adaptive trojaning attacks that are aware of ABS and try to deliberately evade it. We then evaluate ABS on these adaptive attacks to demonstrate its robustness.

The first adaptive attack to bypass ABS is to trojan a model while minimizing the standard deviation of neuron activations in the same layer during data poisoning. The intention is to force multiple neurons to work together to realize the trojaned behavior, by minimizing the activation differences of individual neurons. Ideally, there shall not be a "compromised neuron" to blame.

The second method is to trojan a model such that neuron activations on benign inputs and inputs stamped with triggers are as similar as possible. Since compromised neurons are activated when inputs contain triggers, minimizing the activation difference between benign and malicious inputs has the effect of limiting the elevation effects of compromised neurons such that multiple neurons may be forced to interact to realize the injected behavior.

The third method is to constrain the *maximum* neuron activation differences between benign and malicious inputs. The rationale is similar to the second method.

For the three attacks, we introduce an adaptive loss function in addition to the normal classification loss, and minimize these two loss functions together. For the first adaptive attack, the adaptive loss is the standard deviation of neuron activations within the same layer. For the second attack, the adaptive loss is the mean squared value of activation differences between benign and malicious inputs. For the third attack, the adaptive loss is the maximum mean squared value of activation differences between benign and malicious inputs.

We tune the weight between the normal classification loss and the adaptive loss and obtain a set of trojaned models, with different accuracy on benign inputs and different adaptive loss values. The relation between adaptive loss and model accuracy is shown in Figure 23. The figures in (a), (b), (c) stand for the results for the three attacks: minimized standard deviation, minimized differences between benign and malicious inputs, and minimized maximum differences between benign and malicious inputs, respectively. Each triple (e.g., the three in (a)) includes the results for the NiN, VGG and ResNet110 structures, respectively. The dataset is CIFAR10. In each figure, the x axis is the adaptive loss and the y axis is the model accuracy. The attack success rate (in testing) is always 100% for all these models. As shown in Figure 23, for most trojaned models, along with the decrease of adaptive loss, the model accuracy (on benign inputs) decreases as well. We stop perturbing the weight of adaptive loss when the normal accuracy decreases exceeds 3%. For NiN models and ResNet models of the third type of adaptive loss, we stop perturbing when the adaptive loss is close to 0. For all these trojaned models, ABS can successfully detect them by reverse engineering the top 40 neurons selected through neuron sampling . Recall that without the adaptive attacks, we need to reverse engineer top 10 neurons. The experiment shows that while the adaptive attacks do increase the difficulty, ABS is still quite effective. While more complex and sophisticated adaptive attacks are possible, we will leave such attacks to the future work.

6 DISCUSSION

Although ABS has demonstrated the potential of using an analytic approach and leveraging model internals in detecting trojaned models, it can be improved in the following aspects in the future.

Distinguishing Benign Features from Triggers. As shown in our experiments, ABS occasionally reverse engineers a (strong) benign feature and considers that a trigger. While this is partially true as the feature can subvert other inputs, a possible way to distinguish the two is to develop a technique to check if the reverse engineered "trigger" is present in benign images of the target label.

Handling Complex Feature Space Attacks. In our experiments, we only show two simple feature space attacks. Note that such attacks (for inserting back-doors) have not been studied in the literature as far as we know. As discussed in Section 4.3, complex generative models can be used as the trigger to inject feature space patterns, which may lead to violations of ABS's assumptions and render ABS in-effective. We plan to study more complex feature space attacks and the feasibility of using ABS to detect such attacks.

Handling Label Specific Attacks. Label specific attack aims to subvert inputs of a particular label to the target label. It hence has less persistence. While we have done an experiment to demonstrate that ABS is likely still effective for label specific attack (see Appendix D), we also observe that ABS tends to have more false positives for such attacks and requires additional input samples. We plan to investigate these limitations.

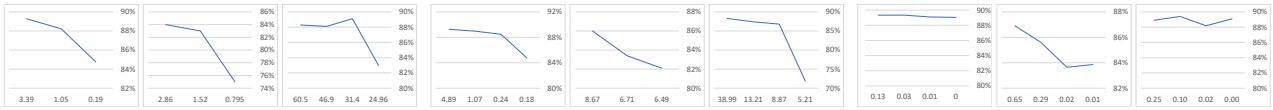
Better Efficiency. Although ABS is much faster than the state-of-the-art, it may need a lot time to scan a complex model. The main reason is that it has to perform the stimulation analysis for every inner neuron. One possibility for improvement is to develop a lightweight method to prune out uninteresting neurons.

One-neuron Assumption. We assume that one compromised neuron is sufficient to disclose the trojan behavior. Although the assumption holds for all the trojaned models we studied, it may not hold when more sophisticated trojaning methods are used such that multiple neurons need to interact to elevate output activation while any single one of them would not. However, as shown in Appendix F, ABS can be extended to operate on multiple neurons. The challenge lies in properly estimating the interacting neurons to avoid exhaustively searching all possible combinations. We will leave it to our future work.

More Relaxed Attack Model. In ABS, we assume that misclassification can be induced by applying trojan trigger on any input. In practice, the attacker may be willing to strike a balance between attack success rate and stealthiness. For example, it may suffice if the attack succeeds on a subset of inputs (say, 80%). In practice, it is also possible that multiple triggers and trigger combinations are used to launch attacks. We leave it to our future work to study ABS's performance in such scenarios.

7 RELATED WORK

In addition to the trojan attacks and defense techniques discussed in Seciton 2, ABS is also related to the following work. Zhou et al. [65] proposed to inject trojan behavior by directly manipulating model weights. However this approach has only been tested on small synthetic model and not yet on real DNNs. Clean label attacks[51, 58] aimed at degrading model performance by poisoning



(a) Min_Std (b) Min_Diff (c) Min_Max_Diff

Figure 23: Model Accuracy (y axis) versus Adaptive Loss (x axis) in Three Adaptive Attacks

the training set with adversarial examples. Human checks cannot recognize such adversarial examples as poisonous but they can change model decision boundary and result in performance degradation. The work in [19, 34] trojans hardware that neural networks are running on. It injects back-door by tampering with the circuits.

Many defense techniques have been proposed to defend against training data poisoning [16, 27, 44, 48]. Most fall into the category of data sanitization where they prune out the poisonous data during training. In contrast, ABS provides defense at a different stage of model life cycle (after they are trained).

ABS is also related to adversarial sample attacks (e.g., [13, 21, 45, 46, 52, 56, 62, 64]). Some recent work tries to construct input-agnostic universal perturbations [43] or universal adversarial patch [15]. These works have a nature similar to Neural Cleanse [59], which we have done thorough comparison with. The difference lies in that ABS is an analytic approach and leverages model internals. A number of defense techniques have been proposed for adversarial examples. Some [18, 40, 57, 63] detect whether an input is an adversarial example and could be potentially used for detecting pixel space trojan triggers. However, these approaches detect inputs with trojan trigger instead of scanning models.

8 CONCLUSIONS

We develop a novel analytic approach to scan neural network models for potential back-doors. It features a stimulation analysis that determines how providing different level of stimulus to an inner neuron impacts model output activation. The analysis is leveraged to identify neurons compromised by trojan attack. Our experiments show that the technique substantially out-performs the state-of-the-art and has a very high detection rate for trojaned models.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by DARPA FA8650-15-C-7562, NSF 1748764, 1901242 and 1910300, ONR N000141410468 and N000141712947, and Sandia National Lab under award 1701331. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] [n.d.]. *BigML Machine Learning Repository.* https://bigml.com/.
[2] [n.d.]. *Caffe Model Zoo.* https://github.com/BVLC/caffe/wiki/Model-Zoo.
[3] [n.d.]. *Gradientzoo: pre-trained neural network models.* https://www.gradientzoo.com/.
[4] 2019. BigML.com. https://bigml.com
[5] 2019. bolunwang/backdoor. https://github.com/bolunwang/backdoor
[6] 2019. BVLC/caffe. https://github.com/BVLC/caffe/wiki/Model-Zoo
[7] 2019. DARPA Announces $2 Billion Campaign to Develop Next Wave of AI Technologies. https://www.darpa.mil/news-events/2018-09-07
[8] 2019. Executive Order on Maintaining American Leadership in Artificial Intelligence. https://www.whitehouse.gov/presidential-actions/executive-order-maintaining-american-leadership-artificial-intelligence
[9] 2019. GitHub - BIGBALLON/cifar-10-cnn: Play deep learning with CIFAR datasets. https://github.com/BIGBALLON/cifar-10-cnn.
[10] 2019. onnx/models. https://github.com/onnx/models
[11] 2019. Tensorpack - Models. http://models.tensorpack.com/.
[12] acoomans. 2013. . https://github.com/acoomans/instagram-filters
[13] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).
[14] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
[15] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial patch. *arXiv preprint arXiv:1712.09665* (2017).
[16] Yinzhi Cao, Alexander Fangxiao Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. 2018. Efficient repair of polluted machine learning systems via causal unlearning. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 735–747.
[17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).
[18] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. 2018. SentiNet: Detecting Physical Attacks Against Deep Learning Systems. *arXiv preprint arXiv:1812.00292* (2018).
[19] Joseph Clements and Yingjie Lao. 2018. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768* (2018).
[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.
[21] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 381–392.
[22] Yarin Gal. 2016. *Uncertainty in deep learning*. Ph.D. Dissertation. PhD thesis, University of Cambridge.
[23] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. *arXiv preprint arXiv:1902.06531* (2019).
[24] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.
[25] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).
[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[27] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–35.
[28] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-reuse attacks on deep learning systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 349–363.
[29] Yujie Ji, Xinyang Zhang, and Ting Wang. 2017. Backdoor attacks against learning systems. In *2017 IEEE Conference on Communications and Network Security (CNS)*.
[30] Melissa King. 2019. TrojAI. https://www.iarpa.gov/index.php?option=com_content&view=article&id=1142&Itemid=443
[31] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.
[32] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).
[33] Gil Levi and Tal Hassner. 2015. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 34–42.

[34] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. 2018. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *ISVLSI*.

[35] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. 2018. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307* (2018).

[36] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 273–294.

[38] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society.

[39] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 45–48.

[40] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. 2019. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *26th Annual Network and Distributed System Security Symposium, NDSS*.

[41] Wei Ma and Jun Lu. 2017. An Equivalence of Fully Connected Layer and Convolutional Layer. *arXiv preprint arXiv:1712.01252* (2017).

[42] Andreas Møgelmose, Dongran Liu, and Mohan M Trivedi. 2014. Traffic sign detection for us roads: Remaining challenges and a case for tracking. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1765–1773.

[44] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. 2015. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics* (2015).

[45] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. ACM, 506–519.

[46] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*.

[47] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. 2015. Deep face recognition.. In *bmvc*, Vol. 1. 6.

[48] Andrea Paudice, Luis Muñoz-González, and Emil C Lupu. 2018. Label sanitization against label flipping poisoning attacks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 5–15.

[49] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[50] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[51] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeuralIPS*.

[52] Mahmood Sharif, Lujo Bauer, and Michael K Reiter. 2018. On the suitability of lp-norms for creating and preventing adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

[53] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[54] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition.. In *IJCNN*, Vol. 6. 7.

[55] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks* 32 (2012), 323–332.

[56] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[57] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. 2018. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In *Advances in Neural Information Processing Systems*. 7717–7728.

[58] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. 2018. Clean-Label Backdoor Attacks. (2018).

[59] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. IEEE, 0.

[60] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[61] wikipedia. 2019. *Electrical brain stimulation - Wikipedia*. https://en.wikipedia.org/wiki/Electrical_brain_stimulation

[62] Xi Wu, Uyeong Jang, Jiefeng Chen, Lingjiao Chen, and Somesh Jha. 2017. Reinforcing adversarial robustness using model confidence induced by adversarial training. *arXiv preprint arXiv:1711.08001* (2017).

[63] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017).

[64] Wei Yang, Deguang Kong, Tao Xie, and Carl A Gunter. 2017. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference*.

[65] Minhui Zou, Yang Shi, Chengliang Wang, Fangyu Li, WenZhan Song, and Yu Wang. 2018. Potrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043* (2018).

Appendix A EXAMPLE OF CONFOUNDING EFFECT

Figure 24 shows a real world example of confounding using a trojaned model on MNIST. As shown in the figure, the NSFs for a benign neuron and a compromised neuron (from the same trojaned model) are similar. Hence, both are candidates. Then we run the model with all the inputs, including the poisonous inputs. We found that the input that induces the largest activation of the benign neuron is a benign input and the largest activation is only 1.8 while a poisonous input can induce the compromised neuron to have a value of 10.4. From the NSF of the benign neuron, there is no elevation effect when the activation is 1.8. The example shows that the elevation effect of a benign neuron usually does not originate from training and may likely be infeasible, whereas the elevation of a compromised neuron is induced by training.

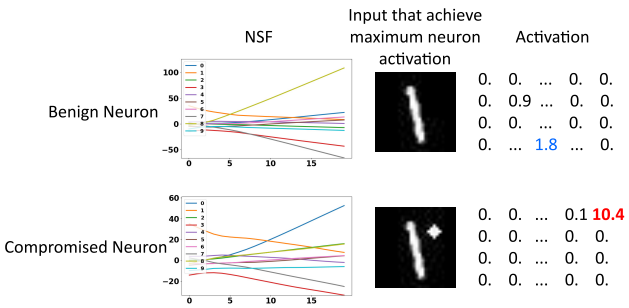


Figure 24: Example of confounding effect

Appendix B MODEL TROJANING

In this section, we show test accuracy and attack success rate of the models trojaned in this paper in Table 7. Columns 1 and 2 show the dataset and models. Column 3 shows the average accuracy of the benign models. Number 20 in the column label indicates that each reported value is an average over 20 models. Column 4 Acc Dec shows the accuracy decrease of models trojaned with trigger YS. Column 5 ASR shows the attack success rate of models trojaned with trigger YS. The accuracy decrease denotes the difference between the accuracy of a benign models and the accuracy of its trojaned version. Note that for CIAR-10 and GTSRB, we trojan 3 models with YS using 3 different combinations of benign and poisonous data, and hence the accuracy decrease and attack success rate are average across 3 models. For ImageNet, we trojan 1 model per trigger, we directly report the accuracy decrease and attack success rate of

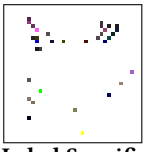


Figure 25: Label Specific "Trigger"

each trojaned model in row 8. ImageNet is a hard task and hence top-5 accuracy is often used [50, 53] . Here we report the top-5 accuracy decrease and the top-5 attack success rate of trojaned models. Columns 6-21 show the accuracy decrease and attack success rate for different kinds of attacks. In some cases, the trojaned models have higher accuracy than the original ones, indicated by negative values in Acc Dec columns. Because the adversarial perturbation is very small, it is hard to trojan an adversarial perturbation model with high attack success rate without degrading the performance on benign images. For adversarial perturbation attack on CIFAR-VGG, the accuracy drops 10.3%. For all other types of attacks, the trojaned models have minor accuracy decrease. The downloaded (benign and trojaned) models have similar test accuracy difference and attack success rate, as shown in their papers [25, 38].

Appendix C DETECTION EFFECTIVENESS VS. TRIGGER SIZE

In this experiment we evaluate how ABS's detection rate changes with trigger size changes on CIFAR-10. In Table 8, the row TS means the size of trigger and column MMS denotes the *max_mask_size* in Algorithm 2. We vary the trigger size from 2% of the input to 25%, following a setting suggested in [30]. As show in Table 8, as long as the *max_mask_size* is larger or equal to the trigger size, the REASR of our approach is at least 85% (100% in most cases). This shows our approach can successfully detect trojaned models within the trigger size budget. In all other experiments, we use $max\_mask\_size = 6\%$.

Appendix D DETECTING LABEL SPECIFIC ATTACK

Although our goal is to defend attacks that aim to persistently subvert any input (of any label) to the target label, we conduct an experiment to observe if ABS has potential in dealing with label specific attack, in which the attacker only aims to subvert inputs of a specific label (called the *victim label*) to the target label. We trojan the CIFAR-10 ResNet model with 10 triggers and each trigger causing the images belonging to class k to be classified to $k + 1$. We poison 10% samples. The model accuracy changes by -0.3% and the average attack success rate (on all labels) is 93.7%. Note that in this context, ABS can no longer use one input per label as stamping inputs of labels other than the victim with the trigger has no effect. Hence, we use 5 additional images for each label for validating if the generated trigger is effective. The REASR scores of benign models and trojaned model are shown in Table 9, with each column (except the first one) representing a label. For benign models, label deer has a high REASR and hence ABS fails. The reverse engineered trigger is shown in Figure 25, which looks like antlers. In general, label specific attack incurs more false positives for ABS because the reverse engineered "trigger" only needs to subvert one class of inputs. For trojaned models, ABS can detect 9 out of 10. ABS fails to detect the trigger that mis-classifies Horse to Ship.

Appendix E PERTURBING PARAMETERS AND USING DATA AUGMENTATION IN NC

In this section, we try different parameter configurations and leverage data augmentation to see if NC's performance can be improved. We perturb 3 main parameters used in NC optimization, the *attack success rate threshold* (THRESHOLD), *cost multiplier* (MULTIPLIER) and *learning rate* (LR). The first is the threshold at which NC stops the optimization and considers the reverse engineered pattern a trigger. The second is used to control the weight between the optimization loss and the trigger size loss. The third is the learning rate for optimization. We experiment different settings of the three parameters on CIFAR10. In addition, since NC's performance is related to the number of samples used in optimization, we try to use data augmentation to increase the number of samples. We use the same data augmentation setting as normal CIFAR-10 training [9]. We test NC with one image per class which is the same setting for ABS. The results are shown in Table 10. Column 1 shows the parameters and column 2 the value settings. Columns 3-5 show the classification accuracy for pixel space trojaned models, feature space trojaned models and benign models for the NiN structure. Columns 6-8 and columns 9-11 show the detection accuracy for VGG and ResNet models. By changing these parameters, the detection rate of pixel space attacks can increase to 60% on VGG models but the accuracy on benign model also decreases to 72%. The detection rate of feature space attacks can increase to 67% on ResNet models but the accuracy on benign model also decreases to 70%. As shown in row 12 in Table 10, using data augmentation increases detection rate on pixel space attacks, while in the mean time causes accuracy degradation on benign models. We speculate that data augmentation can only explore a small region around the input and cannot increase the diversity of data substantially and does not help guide the optimization of NC. This shows changing parameter settings or using data augmentation may improve NC's performance but the improvement is limited.

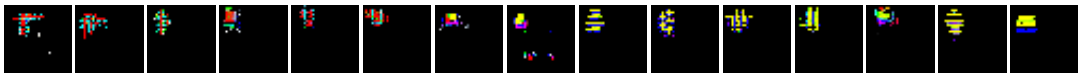
Appendix F OPTIMIZING MULTIPLE COMPROMISED NEURONS TOGETHER

There could be multiple compromised neurons in a trojaned model. In this section, we try to optimize multiple compromised neurons together to see if it can help produce triggers of better quality. For NiN models trojaned with pixel space patch triggers, we further reverse engineer triggers based on all the compromised neurons ABS finds. Figure 26 shows the results. Row 1 shows the original triggers. For each kind of trigger, we trojan the model with 3 different settings such that with the 5 different kinds, there are 15 different models in total. Row 2 shows the triggers reverse engineered using one compromised neuron. ABS may report multiple compromised neurons for a trojaned model. We randomly select one. Row 3 shows the triggers reverse engineered using all compromised neurons together. To use all compromised neurons, we change f_1 in Algorithm 2 to the be sum of the activations of all compromised neurons. As shown in Figure 26, the triggers reverse engineered by multiple neurons are *not* better than those reverse

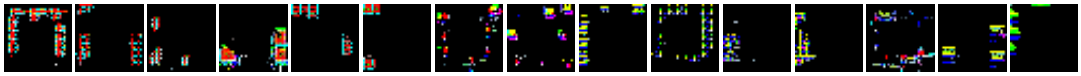
Table 7: Accuracy and Attack Success Rate of Trojaned Models

Dataset	Model	Benign(20)	Pixel										Feature					
			YS(3)		RS(3)		YI(3)		RI(3)		MP(3)		Static(1)		Adversarial(1)		Nashville(3)		Gotham(3)	
		Acc	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR	Acc Dec	ASR
CIFAR-10	NiN	88.7%	1.0%	100.0%	0.6%	99.8%	0.6%	99.8%	4.0%	99.9%	0.4%	100.0%	3.0%	99.9%	4.1%	100.0%	0.9%	100.0%	0.3%	100.0%
	VGG	92.7%	1.8%	100.0%	-0.4%	100.0%	-0.2%	100.0%	-0.5%	100.0%	-0.4%	100.0%	0.1%	98.7%	10.3%	99.4%	-0.4%	100.0%	-0.6%	100.0%
	ResNet32	92.1%	-0.1%	99.9%	-0.1%	99.5%	-0.3%	100.0%	-0.1%	99.5%	-0.3%	100.0%	3.8%	99.3%	1.0%	99.9%	-0.1%	99.7%	-0.1%	100.0%
	ResNet110	93.0%	-0.3%	100.0%	-0.5%	100.0%	-0.5%	100.0%	-0.4%	100.0%	-0.2%	100.0%	1.0%	99.8%	1.0%	96.6%	-0.3%	99.9%	-0.4%	100.0%
GTSRB	NiN	92.4%	-2.2%	100.0%	0.8%	100.0%	-0.1%	100.0%	1.3%	100.0%	0.4%	100.0%	-4.0%	99.9%	-1.3%	98.4%	-2.0%	98.7%	-1.5%	99.7%
	VGG	96.8%	0.1%	100.0%	0.0%	100.0%	0.1%	100.0%	0.1%	100.0%	3.5%	100.0%	0.6%	97.0%	6.1%	97.6%	4.6%	99.7%	4.2%	99.8%
	ResNet110	94.9%	0.7%	100.0%	1.9%	100.0%	2.5%	100.0%	1.2%	100.0%	0.6%	100.0%	2.7%	99.5%	2.6%	100.0%	3.9%	99.3%	1.2%	99.3%
ImageNet	VGG	90.2%	0.0%	92.1%	0.1%	90.5%	0.1%	99.8%	0.0%	92.4%	0.0%	96.6%	1.1%	95.1%	1.1%	99.3%	1.7%	93.8%	1.0%	98.3%

(a) Original Triggers



(b) Triggers Reverse Engineered on 1 Neuron



(c) Triggers Reverse Engineered on Multiple Neurons

Figure 26: Reverse Engineer 1 Neuron vs Multiple Reverse Engineer Neurons

Table 8: REASR of ABS with Different Trigger Size

MMS \ TS	2%	4%	6%	10%	14%	19%	25%
2%	100%	100%	8%	23%	13%	15%	25%
4%	100%	100%	83%	43%	23%	25%	35%
6%	100%	100%	100%	100%	33%	58%	40%
10%	100%	100%	100%	100%	60%	95%	65%
14%	100%	100%	100%	100%	85%	100%	75%
19%	100%	100%	100%	100%	100%	100%	98%
25%	100%	100%	100%	100%	100%	100%	100%

Table 9: REASR for Label Specific Attack

Label	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Trojaned	100%	100%	100%	100%	100%	100%	100%	100%	17%	100%
Benign	17%	33%	33%	17%	100%	50%	17%	0%	0%	17%

Table 10: Perturbing Parameters and Using Data Augmentation in NC

		NiN			VGG			ResNet		
		Pixel	Feature	Benign	Pixel	Feature	Benign	Pixel	Feature	Benign
THRESHOLD	0.8	40%	17%	70%	33%	17%	75%	7%	17%	90%
	0.9	40%	0%	85%	20%	17%	95%	7%	17%	90%
	0.99	40%	17%	85%	27%	17%	100%	7%	17%	85%
MULTIPLIER	1	47%	17%	35%	60%	33%	72%	7%	17%	90%
	2	40%	17%	85%	27%	17%	100%	7%	17%	85%
	4	33%	17%	70%	20%	17%	90%	7%	33%	90%
LR	0.01	47%	0%	75%	13%	17%	95%	7%	17%	90%
	0.1	40%	17%	85%	27%	17%	100%	7%	17%	85%
	1	53%	17%	80%	40%	17%	80%	40%	67%	70%
Data Augmentation		60%	33%	40%	33%	17%	75%	33%	17%	50%

engineered using just a single neuron (from humans' perspective), while both kinds of triggers have very high REASR. This supports our design choice of using one neuron.

Appendix G SENSITIVITY OF PARAMETERS IN ABS

In this section, we evaluate the sensitivity of weight parameters w_1, w_2, w_3 and w_4 in Algorithm 2, where the function cost can be re-written as follows:

$$cost = w_1(\cdot f_2 - \frac{w_2}{w_1} \cdot f_1 + \frac{w_3}{w_1} \cdot sum(mask) - \frac{w_4}{w_1} \cdot SSIM(x, x'))$$

Note that w_1 is a common parameter for all entries in $cost$. Thus we can focus on tuning w_2, w_3 and w_4 to show the sensitivity of parameters in ABS. Recall that w_3 is the parameter for pixel space triggers and w_4 is the parameter for feature space triggers. Hence we perturb w_2 and w_3 for pixel space attacks, and w_2 and w_4 for feature space attacks. We perturbed these parameters on the trojaned NiN models at a scale of 10. The default value of w_2 is 1e-4 and we try different values of w_2 ranging from 1e-6 to 1e-2. The default value of w_3 is 50 and we try different values from 0.5 to 500. The default value of w_4 is 10 and hence we try the values from 0.1 to 1000. In Table 11, row 1 shows the different parameters, row 2 shows the different values of parameters and row 3 shows the average REASR for the trojaned NiN models. Observe that changing w_2 and w_3 does not significantly impact REASR. For w_4, ABS is effective from 0.1 to 10, but has degraded performance beyond 10.

Table 11: Sensitivity of ABS parameters

w_2					w_3					w_4				
1E-06	1E-05	1E-04	1E-03	1E-02	0.5	5	50	500	5000	0.1	1	10	100	1000
0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.25	0.1