Empirical defenses against backdoor attacks

Presented by
Haoming Yang, Rucha Patil, Oded Schlesinger, Angikar Ghoshal

Index

1. Background
2. Detection STRIP
3. Detection in Neural Cleanse , Identification
4. Experiments on Detection, Identification
5. Mitigation
6. Mitigation Experiments
7. Limitation/Future Work

DNNs

- Play an important role in wide range of critical applications
- Lack of interpretability
- Numerical blackboxes : inability to test exhaustively
- Computationally Expensive : Outsourced

What are Backdoor Trojans

- Backdoors are hidden patterns that have been trained into a DNN model that produce unexpected behavior, but are undetectable unless activated by some “trigger” input.
- Attack is simple, robust, highly effective, easy to realize
- In other Adversarial examples the attacker does not have full control over converting the physical scene into an effective adversarial digital input
- Vision Systems : input agnostic trigger
- Insertion can take place during Training Phase or the Tuning Phase
- 99% successful on the MNIST dataset

Challenges in Detection

- Defender has no knowledge of Trigger
- Trigger features
  - Arbitrary shape and pattern
  - Located in any position
  - Any Size
- Insertion can take place
  - During Training Phase
  - During Tuning Phase
- No way to validate anomalous training data
- Can be inconspicuous

Mathematically Representing DNNs

- A DNN is a parameterized function F_θ that maps a n-dimensional input $x \in \mathbb{R}^n$ into one of M classes.
- The y_i is the probability of the input belonging to class (label) i.
- An input x is deemed as class i with the highest probability such that the output class label z is $argmax_{i\epsilon[1;M]}\, y_i$.
- Training Dataset : $D_{train} = \{xi, yi\}_{i=1}^{S}$ of S inputs
  Where, $x \in \mathbb{R}^N$ and corresponding ground-truth labels , $z_i \in [1, M]$

$$\Theta = \underset{\Theta^*}{\operatorname{argmin}} \sum_{i}^{S} \mathcal{L}(F_{\Theta^*}(x_i), z_i).$$

- Validation Dataset : $D_{valid} = \{xi, yi\}_{1}^{V}$ with V inputs

Mathematically Representing Trojans

- Given a benign input xi, the prediction $\tilde{y}\imath = F_\theta(xi)$ of the trojaned model has a very high probability to be the same as the ground-truth label yi.
- Given a trojaned input $xai = xi + xa$, the predicted label will always be the class za set by the attacker, regardless of what the specific input xi is.

Is there an inherent weakness in trojan attacks with input-agnostic triggers that is easily exploitable by the victim for defence?

Consider an Example

- A trojan is inserted in the MNIST dataset with trojan accuracy of 99.86%
- 600 poisoned samples out of 50000, clean input accuracy of 98.86%
- Trojan is input-agnostic

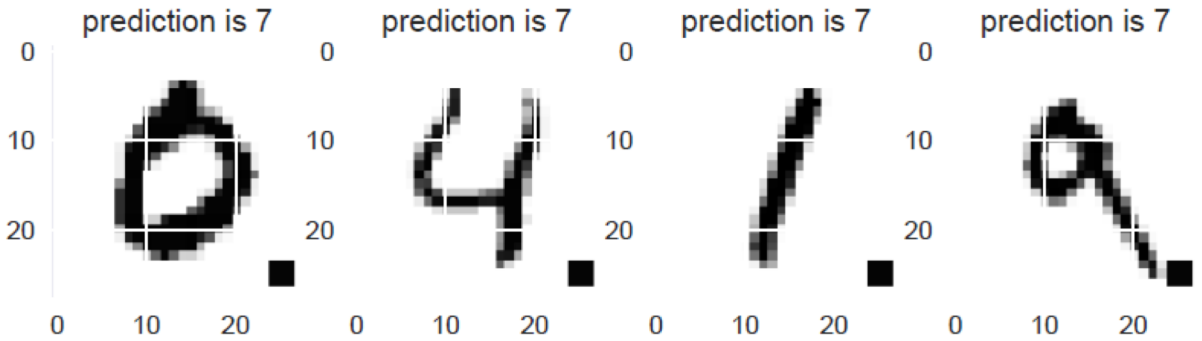


Figure 2. Trojan attacks exhibit an input-agnostic behavior. The attacker targeted class is 7.

The STRIP Detection Example - 1

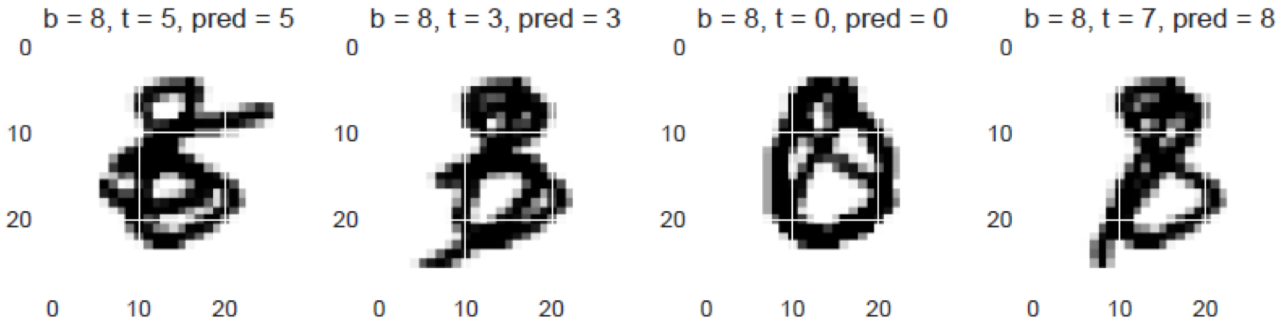


Figure 3. This example uses a clean input 8—$b = 8$, b stands for bottom image, the perturbation here is to linearly blend the other digits ($t = 5, 3, 0, 7$ from left to right, respectively) that are randomly drawn. Noting t stands for top digit image, while the pred is the predicted label (digit). Predictions are quite different for perturbed clean input 8.

The STRIP Detection Example - 2

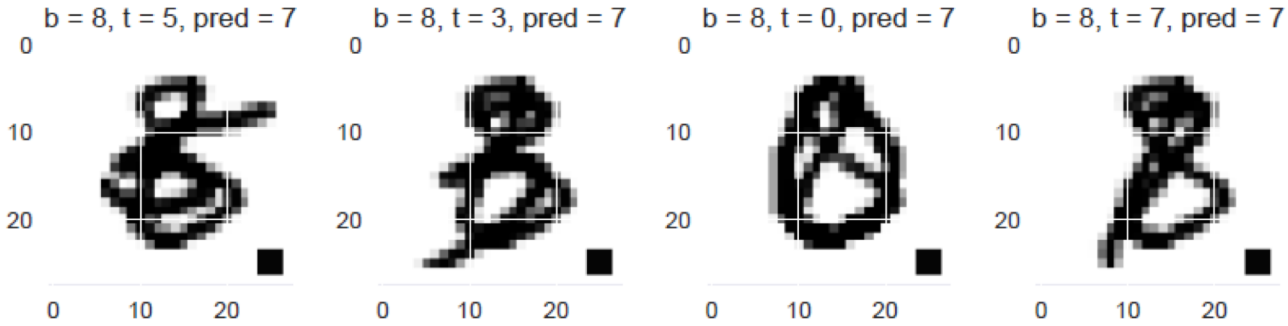


Figure 4. The same input digit 8 as in Fig. 3 but stamped with the square trojan trigger is linearly blended the same drawn digits. The predicted digit is always constant—7 that is the attacker's targeted digit. Such constant predictions can only occur when the model has been malicious trojaned and the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

The STRIP Detection Example - 3

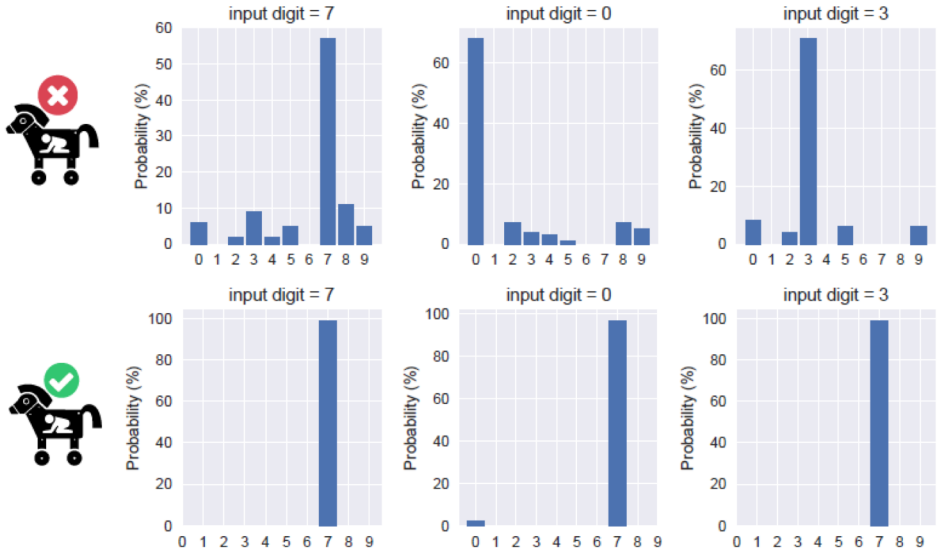
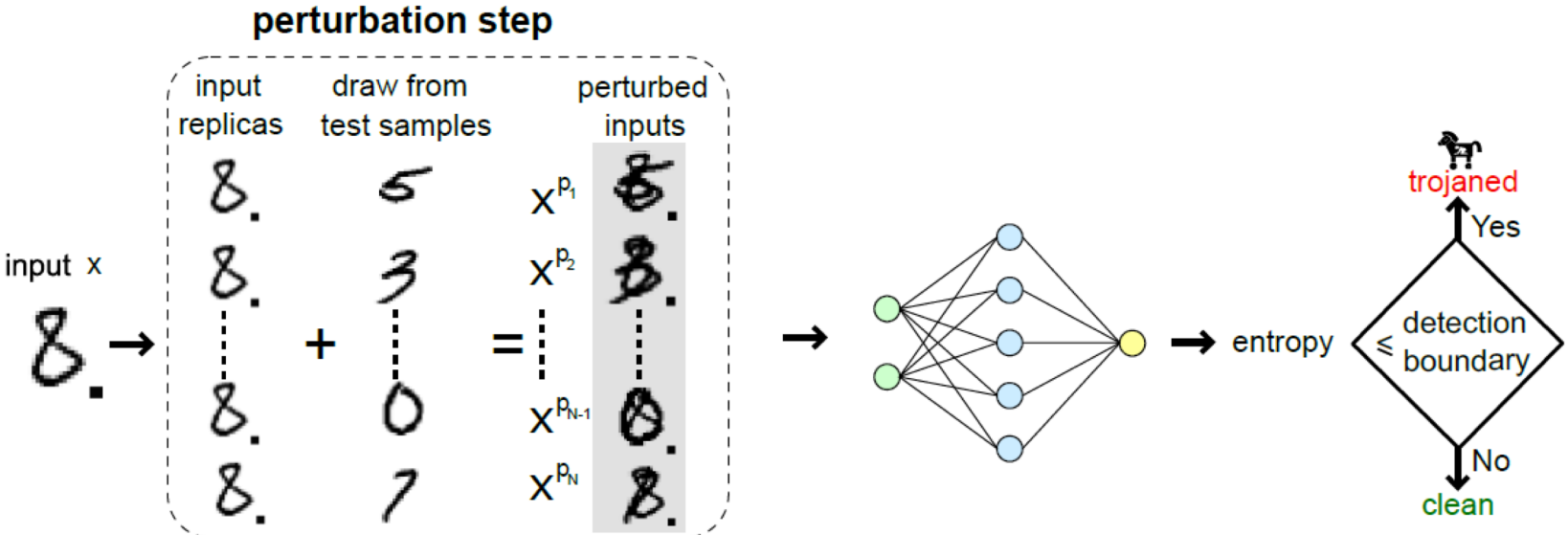


Figure 5. Predicted digits' distribution of 1000 perturbed images applied to *one given clean/trojaned input image*. Inputs of top three sub-figures are trojan-free. Inputs of bottom sub-figures are trojaned. The attacker targeted class is 7.

The **STR**ong **I**ntentional **P**erturbation



Detection System Algorithm

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

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

Threat Model

- Capabilities of the Attacker
  - Full access to training dataset
  - Full access to white-box to DNN model/architecture
  - Attacker can determine pattern, location and size of the trigger
- Capabilities of the Defender
  - Defender has held out a small collection of validation samples
  - Does not have access to trojaned data stamped with triggers

Metrics

- FRR : False Rejection Rate

  Probability when benign input is regarded as trojan input by the STRIP Detection System

- FAR : False Acceptance Rate

  Probability when trojan input is regarded as benign input by the STRIP Detection System

Ideally, both the values should be 0%

Entropy

Shannon Entropy

The H is regarded as the entropy of one incoming input x. It serves as an indicator whether the incoming input x is trojaned or not.

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

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

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

Table I
DETAILS OF MODEL ARCHITECTURE AND DATASET.

Dataset	# of labels	Image size	# of images	Model architecture	Total parameters
MNIST	10	$28 \times 28 \times 1$	60,000	2 Conv + 2 Dense	80,758
CIFAR10	10	$32 \times 32 \times 3$	60,000	8 Conv + 3 Pool + 3 Dropout 1 Flatten + 1 Dense	308,394
GTSRB	43	$32 \times 32 \times 3$	51,839	ResNet20 [25]	276,587

The GTSRB image is resized to $32 \times 32 \times 3$.

Advantages and Limitations of STRIP

- Advantages
  - Plug and Play approach, Compatible with existing DNN
  - Independent of deployed DNN model architecture
  - Insensitive to the trigger-size employed
  - Achieves 0% FAR and FRR in most tested cases
- Limitations
  - Lacking generalization currently
  - Ineffective on source-label specific triggers
  - Defender does not have access to trojan samples in real life

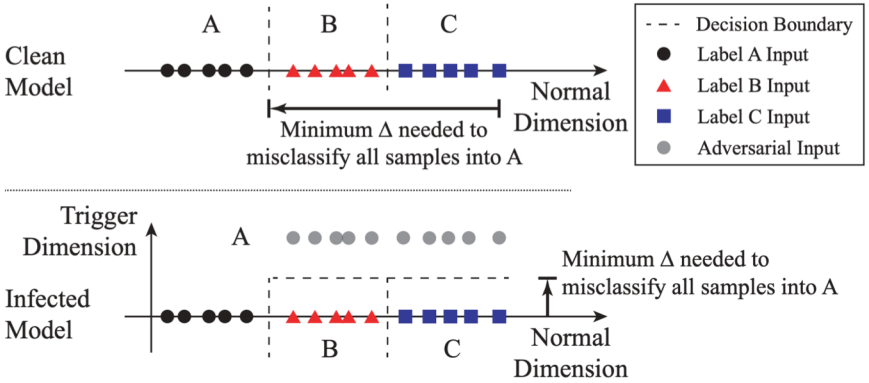
Neural Cleanse Detection (and Identification)

A Key Insight:

- From the Classification POV: Model aims to create partitions that separates latent space.
- From Infected Model POV: Infected model has partition shortcut.
- From Backdoor Attack POV: Clean samples' latent representation are correctly classified. Triggers acts as a perturbation to nudge the latent representation to target partition through the shortcut.

Neural Cleanse Aims to reverse engineer this behavior

Neural Cleanse Detection (and Identification)



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

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

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

In human terms:

If a model is backdoored, it requires abnormally low value of perturbation (comparing to others) to transition a classification result from an arbitrary label to the target label

Neural Cleanse Detection (and Identification)

Step 1:

Given a Label, treated as the potential target.

Optimize to find the “minimal” trigger to misclassify all samples from other labels into this target label.

Step 2:

Repeat Step 1 for all Labels.

Collect all found “minimal” trigger.

Step 3:

Find the outliers of “minimal” triggers, the one that is abnormally minimal.

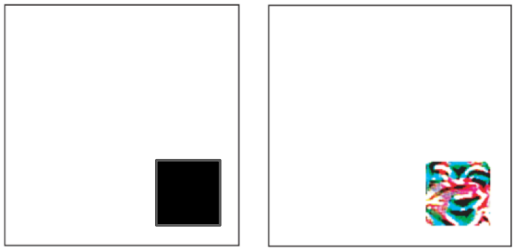
Note 1*: If a trigger is found in Step 3, it automatically satisfies the identification of trigger
Note 2*: Step 2 is obviously computational intensive, an algorithm was proposed to alleviate the issue

Neural Cleanse Detection (and Identification)

Represent a Trigger

$$A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) = \boldsymbol{x'}$$

$$\boldsymbol{x'}_{i,j,c} = (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c}$$



Formalize Optimization

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

Neural Cleanse Detection (and Identification)

- Anomaly Detection using Median Absolute Deviation

$$\mathrm{MAD} = \mathrm{median}(|X_i - \tilde{X}|)$$

$$\tilde{X} = \mathrm{median}(X)$$

Eg.
Data: (1,1,2,**2**,4,6,9), Absolute Deviation: (1,1,0,0,2,4,7), Potential Outlier: 7

Neural Cleanse Detection (and Identification)

- An obvious problem: Computational efficiency
- Leading to Low Cost algorithm:
- The optimization mostly converge in the first few iterations, and fine tunes the found “minimal” trigger for the rest of the iterations.
- Stop Early -> Narrow down the potential target -> Fine Tune

Detection Experiments (STRIP)

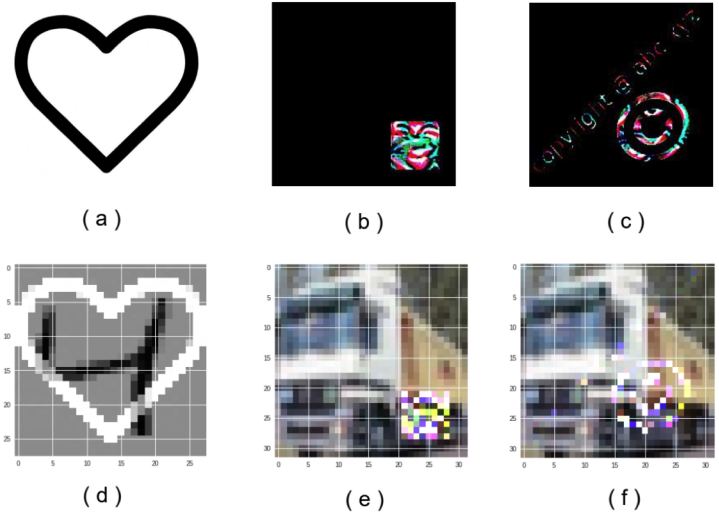


Figure 7. Besides the square trigger shown in Fig. 2. Other triggers (top) identified in [16], [17] are also tested. Bottom are their corresponding trojaned samples.

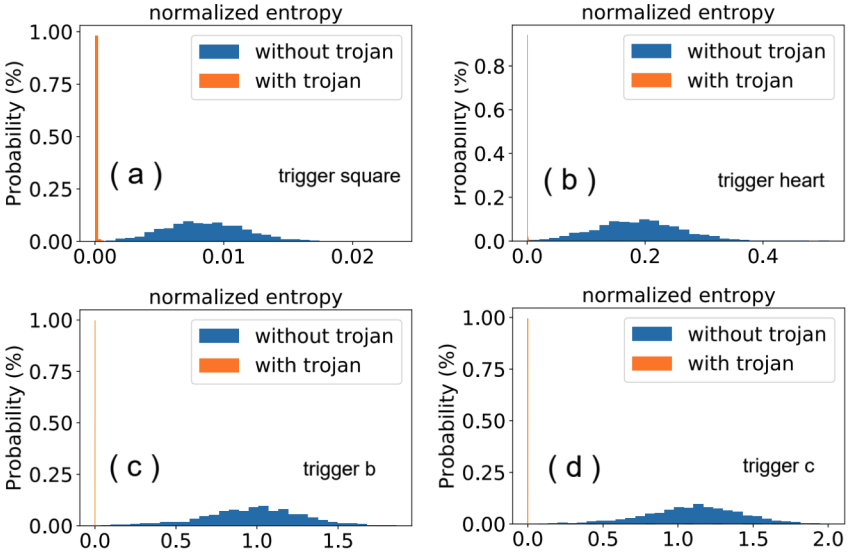


Figure 8. Entropy distribution of benign and trojaned inputs. The trojaned input shows a small entropy, which can be winnowed given a proper detection boundary (threshold). Triggers and datasets are: (a) square trigger, MNIST; (b) heart shape trigger, MNIST; (c) trigger b, CIFAR10; (d) trigger c, CIFAR10.

Detection Experiments (STRIP)

FAR AND FRR OF STRIP TROJAN DETECTION SYSTEM.

Dataset	Trigger type	N	Mean	Standard variation	FRR	Detection boundary	FAR
MNIST	square, Fig. 2	100	0.196	0.074	3% 2% 1%[1]	0.058 0.046 0.026	0.75% 1.1% 1.85%
MNIST	trigger a, Fig. 7 (a)	100	0.189	0.071	2% 1% 0.5%	0.055 0.0235 0.0057	0% 0% 1.5%
CIFAR10	trigger b, Fig. 7 (b)	100	0.97	0.30	2% 1% 0.5%	0.36 0.28 0.20	0% 0% 0%
CIFAR10	trigger c, Fig. 7 (c)	100	1.11	0.31	2% 1% 0.5%	0.46 0.38 0.30	0% 0% 0%
GTSRB	trigger b, Fig. 7 (b)	100	0.53	0.19	2% 1% 0.5%	0.133 0.081 0.034	0% 0% 0%

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

Note: In appendix B of STRIP, the author also mentioned that detection capability improves with depth of neural networks

Detection Experiments (Neural Cleanse)

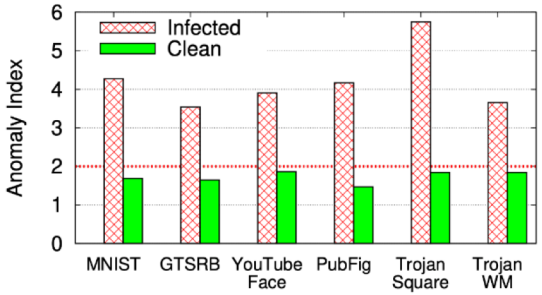


Fig. 3. Anomaly measurement of infected and clean model by how much the label with smallest trigger deviates from the remaining labels.

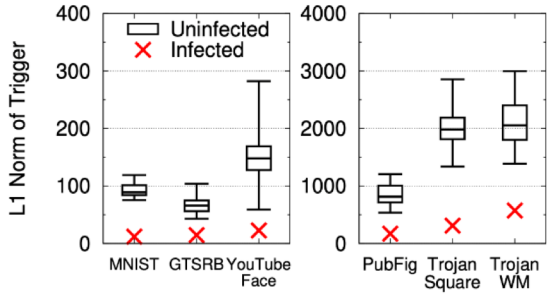


Fig. 4. $L1$ norm of triggers for infected and uninfected labels in backdoored models. Box plot shows min/max and quartiles.

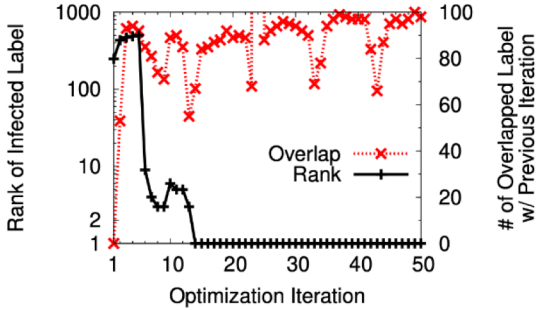
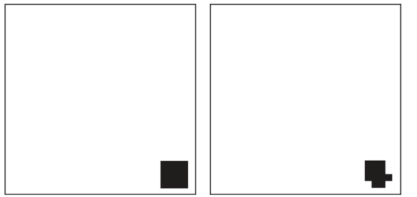


Fig. 5. Rank of infected labels in each iteration based on trigger's norm. Ranking consistency measured by # of overlapped label between iterations.

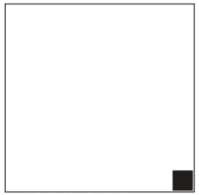
Detection Experiments (Neural Cleanse)



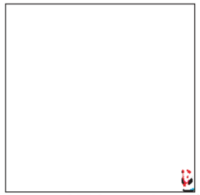
Original Trigger (L1 norm = 16)

Reversed Trigger ($m \cdot \Delta$) (L1 norm = 12.00)

(a) MNIST



Original Trigger (L1 norm = 576)



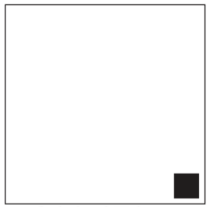
Reversed Trigger ($m \cdot \Delta$) (L1 norm = 171.11)

(d) PubFig

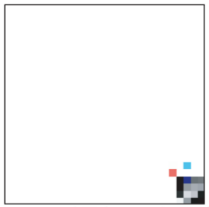
Original Trigger (L1 norm = 3,481)

Reversed Trigger (m) (L1 norm = 311.24)

(a) Trojan Square

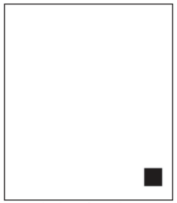


Original Trigger (L1 norm = 16)



Reversed Trigger ($m \cdot \Delta$) (L1 norm = 14.71)

(b) GTSRB

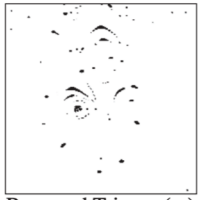


Original Trigger (L1 norm = 25)

Reversed Trigger ($m \cdot \Delta$) (L1 norm = 22.79)

(c) YouTube Face

Original Trigger (L1 norm = 3,598)



Reversed Trigger (m) (L1 norm = 574.24)

(b) Trojan Watermark

Detection Experiments (Neural Cleanse)

Neural Activation Similarity

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

Mitigation of Backdoors

Two techniques:

- Creating a filter for adversarial input that identifies and rejects any input with the trigger, giving us time to patch the model
- Patching the DNN, making it nonresponsive against the detected backdoor triggers
    - Neuron Pruning
    - Unlearning

Filter for Detecting Adversarial Inputs

- **Idea:**
  Neuron activations are a better way to capture similarity between original and reverse-engineered triggers.
- **Technique:**
  Given some input, the filter identifies potential adversarial inputs as those with activation profiles higher than a certain threshold. The activation threshold can be calibrated using tests on clean inputs.
- **Evaluation:**
  High filtering performance for all four BadNets models, with < 1.63% FNR at an FPR of 5%. TrojanAttack models are more difficult to filter out (likely due to the differences in neuron activations between reversed trigger and original trigger). FNR is much higher for FPR < 5%, but we obtain a reasonable 4.3% and 28.5% FNR at an FPR of 5%.

Patching via DNN Pruning

- **Idea:**
  Use the reversed trigger to help identify backdoor related components in DNN, e.g., neurons, and remove them.

- **Technique:**
  Target the second to last layer, and prune neurons by order of highest rank first (i.e. prioritizing those that show biggest activation gap between clean and adversarial inputs). To minimize impact on classification accuracy of clean inputs, stop pruning when the pruned model is no longer responsive to the reversed trigger.

- **Observation:**
  For GTSRB, attack success rate of the reversed trigger follows a similar trend as the original trigger, and thus serves as a good signal to approximate defense effectiveness to the original trigger. Pruning 30% of neurons reduces classification accuracy by 5% but attack success rate goes to almost 0%. (massive redundancy!)

Patching via DNN Pruning

Evaluation:

- For YouTube Face, classification accuracy drops from 97.55% to 81.4% when attack success rate drops to 1.6%. (Only 160 neurons in second to last layer, so clean neurons get pruned too). Pruning at the last convolution layer produces the best results.
- In all four BadNets models, attack success rate reduces to < 1% with minimal reduction in classification accuracy < 0.8%. Meanwhile, at most 8% of neurons are pruned.
- For Trojan models, when pruning 30% neurons, attack success rate using reverse-engineered trigger drops to 10.1%, but success using the original trigger remains high, at 87.3%. This discrepancy is due to the dissimilarity in neuron activations between reversed trigger and the original trigger.

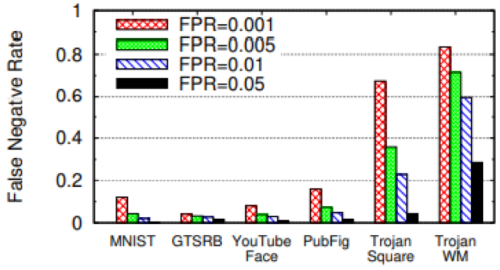


Fig. 8. False negative rate of proactive adversarial image detection when achieving different false positive rate.

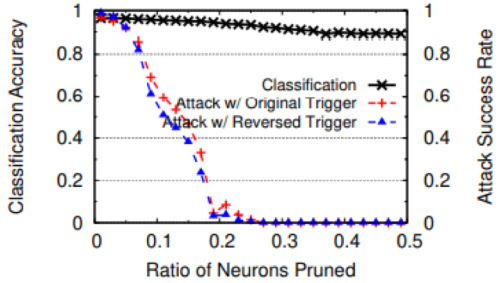


Fig. 9. Classification accuracy and attack success rate when pruning trigger-related neurons in GTSRB (traffic sign recognition w/ 43 labels).

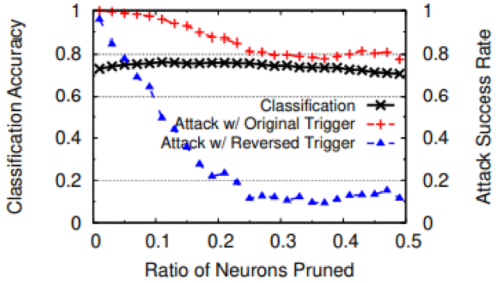


Fig. 10. Classification accuracy and attack success rate when pruning trigger-related neurons in Trojan Square (face recognition w/ 2,622 labels).

Patching DNNs via Unlearning

- **Idea:**

  Use the reversed trigger to train the infected DNN to recognize correct labels even when the trigger is present.

- **Technique:**

  Fine-tune the model for only 1 epoch, using an updated training dataset. To create this new training set, we take a 10% sample of the original training data (clean, with no triggers), and add 7% (or full training data if training data is very limited), and add the reversed trigger to 20% of this sample without modifying labels.

Patching DNNs via Unlearning

Results:

- In all models, attack success rate is reduced to < 6.70%, without significantly sacrificing classification accuracy.
- The largest reduction of classification accuracy is in GTSRB, which is only 3.6%.
- For some models, especially Trojan Attack models, there is an increase in classification accuracy after patching.
- When injecting the backdoor, the Trojan Attack models suffer degradation in classification accuracy. Original uninfected Trojan Attack models have a classification accuracy of 77.2% which is now improved when the backdoor is patched.

Patching DNNs via Unlearning

Comparisons:

- Training against the same training sample, but applying the original trigger instead of the reverse-engineered trigger

Unlearning using the original trigger achieves slightly lower attacker success rate with similar classification accuracy. So unlearning with our reversed trigger is a good approximation for unlearning using the original.

- Unlearning using only clean training data (no additional triggers)

Unlearning is ineffective for all BadNets models (attack success rate still high) but highly effective for Trojan Attacks.

BadNets V Trojan Attacks

- Trojan Attack models, with their highly targeted re-tuning of specific neurons, are much more sensitive to unlearning. A clean input that helps reset a few key neurons disables the attack.
- In contrast, BadNets injects backdoors by updating all layers using a poisoned dataset, and requires significantly more work to retrain and mitigate the backdoor.
- Even though unlearning has a higher computational cost compared to neuron pruning, it is still one to two orders of magnitude smaller than retraining the model from scratch.

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

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

Robustness against Advanced Backdoors

- Study more advanced variants of backdoor attacks and evaluate their impact on their proposed techniques.
- Further investigate and evaluate the performance over several types of advanced backdoor attacks, among them:
  - Different Trigger Shapes
  - Larger Triggers
  - Multiple Infected Labels with Separate Triggers
  - Single Infected Label with Multiple Triggers

Different trigger shapes

- As observed (Trojan square and watermark), triggers with more complicated patterns make it harder for the optimization process to converge correctly.
- How would other triggers (of the same size) affect the convergence of the optimization function used for detection?

Different trigger shapes

- As observed (Trojan square and watermark), triggers with more complicated patterns make it harder for the optimization process to converge correctly.
- How would other triggers (of the same size) affect the convergence of the optimization function used for detection?
- Neural Cleanse authors perform simple tests by:
  - noisy squares
  - different trigger shapes
- On MNIST, GTSRB, YouTube Face, PubFig, the proposed techniques still works.

Larger Triggers

- Larger triggers are likely to produce larger reverse engineered triggers. Are they easier or more difficult to detect?
- Experiments to evaluate the trigger size effect, increasing the size of trigger from 4×4 (1.6% of the image) to 16×16 (25%) on GTSRB dataset.

Larger Triggers

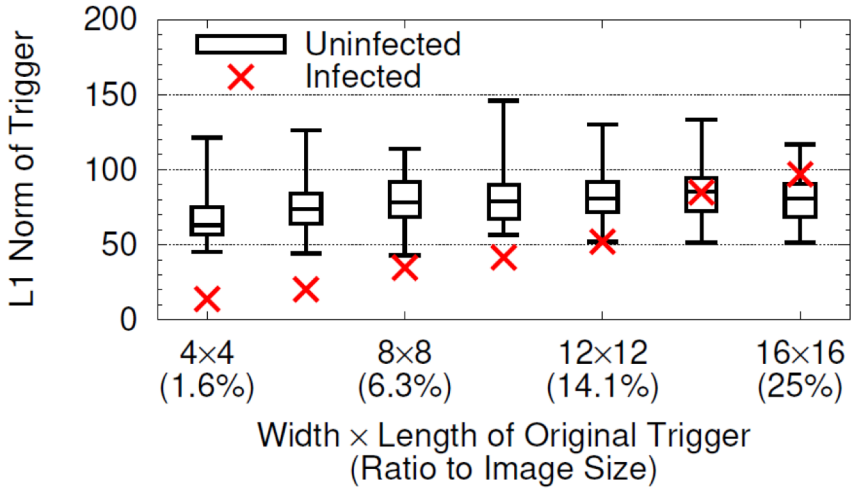


Fig. 12. $L1$ norm of reverse engineered triggers of labels when increasing the size of the original trigger in GTSRB (results of a single round).

Larger Triggers

- Setting the transparency of a Hello Kitty image to 70% over an entire CIFAR10 input image.
- The evaluated minimal entropy of clean images is 0.0035 and the maximal entropy of trojan images is 0.0024.
- STRIP managed to detect the backdoor attack.

Multiple Infected Labels with Separate Triggers

- Inserting multiple, independent backdoors into a single model, each infecting another label.
- Evaluating the maximum number of infected labels the proposed defense is able detect effectively.
- Reduces the change required to get to different target labels - might cause outliers to be harder to detect.

Multiple Infected Labels with Separate Triggers

- Generating unique triggers with exclusive color patterns for different target labels.
- Evaluating the defense methods against them in GTSRB.

Multiple Infected Labels with Separate Triggers

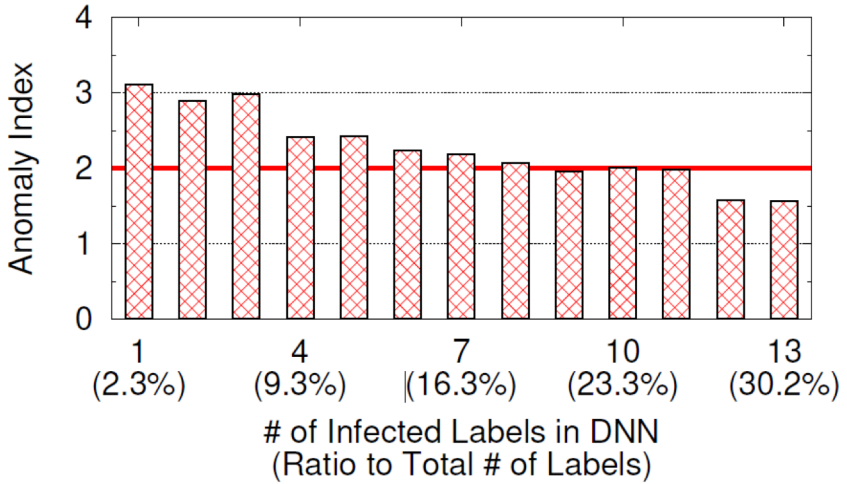


Fig. 15. Anomaly index of each infected GTSRB model with different number of labels being infected (results averaged over 10 rounds).

Multiple Infected Labels with Separate Triggers

- Similar results in other datasets:
  - 3 labels (30%) for MNIST
  - 375 labels (29.2%) for YouTube Face
  - 24 labels (36.9%) for PubFig

Single Infected Label with Multiple Triggers

- Consider various triggers that induce misclassification to the same infected label.
- Since the identification task is formulated as an optimization problem, it will converge into one trigger.
- Neural Cleanse authors inject 9 white square triggers of the original size for a single GTSRB label.

Single Infected Label with Multiple Triggers

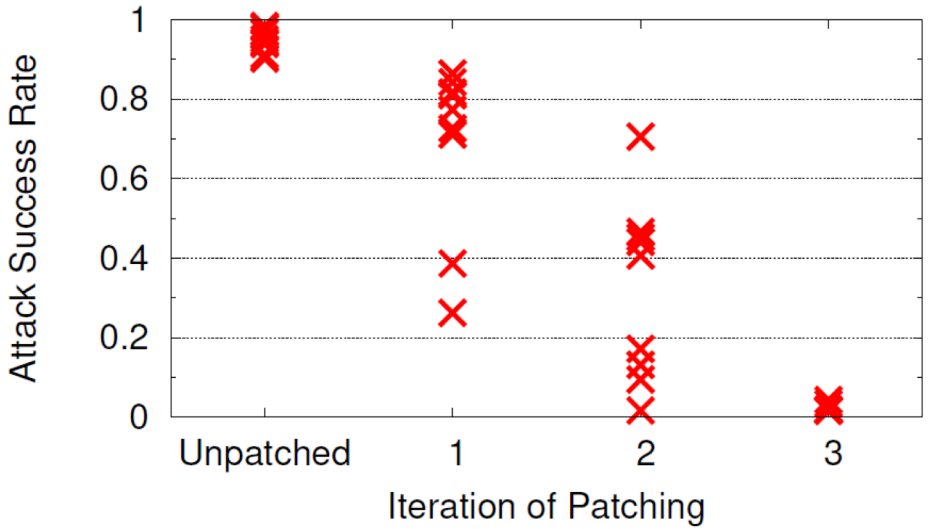


Fig. 17. Attack success rate of 9 triggers when patching DNN for different number of iterations.

Single Infected Label with Multiple Triggers

- Similar results in other datasets:
  - <1% for MNIST
  - <5% for YouTube Face
  - <4% for PubFig
- In contrast, according to the STRIP authors' evaluation with CIFAR10, where each trigger is a small digit (size is not mentioned), they achieve perfect detection with no false rejection at all.

Limitations and Future Work

- Both:
  - Source-label specific triggers
- Neural Cleanse:
  - Relies on outlier detection, vulnerable to numerous targeted labels
  - The neuron pruning approach performance depends on choosing the right layer which requires expertise
- Other types of data (graphs, audio etc.).