CYBER SECURITY PROJECT REPORT

CSAW-HackML-2020

Shiwangi Mishra (SM9175)

GitHub Repository: <https://github.com/ShiwangiMishra-Git/ML-CyberSecurtiyProject/tree/master/ML-CyberSecurity-Project>

Environment Setup

Please check the README in the GitHub repository. We introduce the dependencies as well as the bash

commands to run the code.

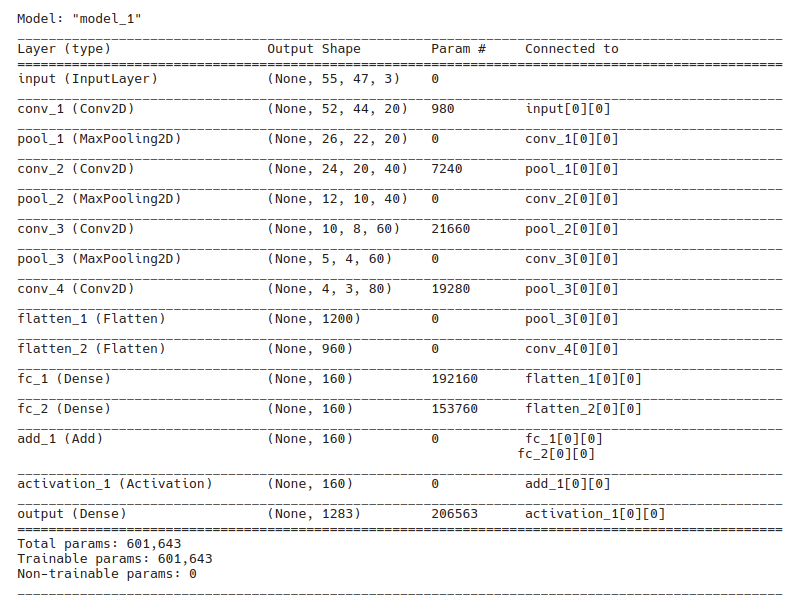
Base Model Structure

We implemented two approaches to detect and repair backdoored model. First is based on by Neural Cleanse

and the second is based on STRIP. Both approaches can be divided into two parts separately, e.g. detecting the backdoor labels

and repair the BadNets. Our models are based on the default model in the origin repo, as following Fig. 1

Shows.



Neural Cleanse

Detect Backdoors

The key idea of Neural Cleanse by detecting backdoors is that if a model is poisoned, it requires much smaller modifications

to cause the model to classify the wrong target label. So we decided to iterate all possible labels and check

which one requires smaller modification to achieve the wrong result. The whole process will be divided into

3 steps:

1. Find the minimal trigger. We try to find a trigger window with a fixed label. We assume this label is

the target label of the attack backdoor trigger. The performance of this trigger depends on how small

it is to misclassify all samples from other labels into the target label.

2. Iterate the whole label sets. We run the loop for iterating all labels in the model, which is 1283 in our

project. In other words, 1283 potential triggers will be created after this step.

3. Choose the valid trigger. We need to choose the valid trigger in all 1283 triggers. It depends on the

number of pixels the trigger trying to influence in the models. Our method is to calculate the L1 norms

of all triggers. Then we will calculate the absolute deviation between all data points and the median.

If the absolute deviation of a data point divided by that median is larger than 2, we mark it as a target

trigger. The target trigger which is most effective to misclassify the model will be the “reverse trigger”

we need to repair BadNets.

The implementation of the step 1 and 2 is in the visualize\_example.py and visualizer.py.

The implementation of the step 3 is in the mad\_outlier\_detection.py.

Repair BadNets

In order to repair BadNets, we decided to patch the infected model by pruning the poisoned neurons in the

BadNet with the “reverse trigger”.

The target trigger poisoned neurons in the model to make it misclassify the label, so we need to find these

neurons and set their output value to 0 so that the model will not be affected by the trigger anymore.

Therefore we rank the neurons by differences between clean input and poisoned input produced by the ’reverse

triggers’. We again target the second to last layer, and prune neurons by order of highest rank first. In order

to keep the performance of the model on clean target, we decided to stop the iteration as soon as the model

is not sensitive to the poisoned input any more.

You can find details in the repair\_model.py.

Result & Sample Output

#!/bin/bash

python3 repair\_model.py sunglasses

base model in clean test: 97.77864380358535, poisoned: 99.99220576773187

pruned model in clean test: 86.83554169914264, poisoned: 1.161340607950117

repair model in clean test: 88.08261886204208, fixed poisoned: 100.0

elapsed time 90.44689536094666 s

python3 repair\_model.py anonymous\_1

base model in clean test: 97.1862821512081, poisoned: 91.3971161340608

pruned model in clean test: 95.12081060015588, poisoned: 3.0982073265783323

repair model in clean test: 79.81293842556508, fixed poisoned: 99.71745908028059

elapsed time 67.95545625686646 s

python3 repair\_model.py anonymous\_2

base model in clean test: 95.96258768511302, poisoned: 0.0

pruned model in clean test: 96.18862042088854, poisoned: 0.03897116134060795

repair model in clean test: 78.95557287607171, fixed poisoned: 99.85385814497272

elapsed time 60.89538335800171 s

python3 repair\_model.py multi\_trigger\_multi\_target

base model in clean test: 96.00935307872174, poisoned: 30.452714990906728

pruned model in clean test: 95.86905689789556, poisoned: 1.575084437516238

repair model skipped.

elapsed time 150.934408903122 s

STRIP

Introduction

STRIP is a run-time trojan detection system that can distinguish trojaned input from clean ones.

Principle

The principles of STRIP can be illustrated by an example on MNIST handwritten digits. As attack shown in

Fig. 2, the trigger is a square located at the bottom-right corner and the target of the attackers is 7. For a

trojaned input, the predicted digit is always 7 that is what the attacker wants - regardless of the actual input

digit — as long as the square at the bottom-right is stamped.

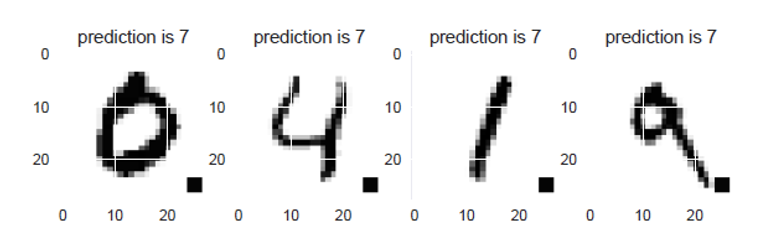


Figure 2: Trojan attacks exhibit an input-agnostic behavior. The attacker targeted class is 7.

This input-agnostic characteristic is recognized as main strength of the trojan attack which is exploitable to

detect whether a trojan trigger is contained in the input from the perspective of a defender. The key insight

is that, regardless of strong perturbations on the input image, the predictions of all perturbed inputs tend

to be always consistent, falling into the attacker’s targeted class. This behavior is eventually abnormal and

suspicious. Because, given a benign model, the predicted classes of these perturbed inputs should vary, which

strongly depend on how the input is altered. Therefore, we can intentionally perform strong perturbations

to the input to infer whether the input is trojaned or not.

In Fig. 4, the input is 8 and is clean. The image linear blend perturbation here is superimposing two images.

The digit images to be perturbed with clean input are randomly drawn. Each of the drawn digit image is

then linearly blended with the incoming input image. We expect the predicted numbers (labels) of perturbed

inputs should vary significantly since such strong perturbations on the benign input should greatly influence

its predicted label and the randomness of selection of images to be perturbed ensure the results are unpredictable.

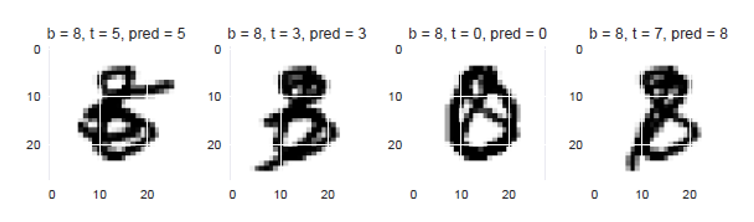


Figure 3: This example uses a clean input 8 - b = 8, b stands for bottom image, the perturbation here is to

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 perturbation strategy works well on clean input. But for the trojaned input, the predicted labels are dominated

by the trigger, regardless of the influence of strong perturbation. As shown in Fig. 4, the prediction

are Surprisingly consistent. Such an abnormal behavior violates the fact that the model prediction should be

input-dependent for a benign model. So we can conclude that the input is trojaned, and the model under

deployment is very likely backdoored.

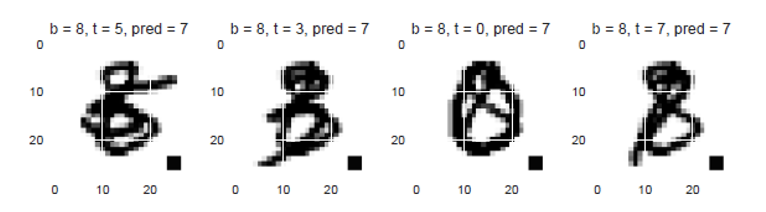


Figure 4: The same input digit 8 as in Fig. but stamped with the square trojan trigger is linearly blended the

same drawn digits. Such constant predictions can only occur when the model has been malicious trojaned

and the input also possesses the trigger.

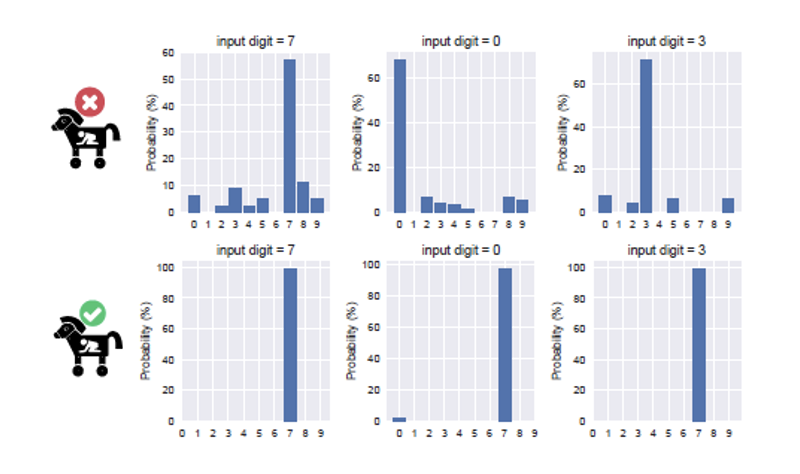


Figure 5: Predicted digits’ distribution of 1000 perturbed images applied to one given clean/trojaned input

image. Inputs of top three sub-figures are trojan-free. Input of bottom sub-figures are trojaned. The attacker

targeted class is 7

This figure depicts the predicted classes’ distribution given that 1000 randomly drawn digit images are linearly

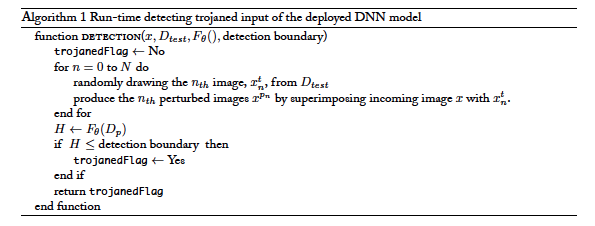
blended with one given incoming benign and trojaned input, respectively. Overall, high randomness of

predicted classes of perturbed inputs implies a benign input; whereas low randomness implies a trojaned

input.

Method

To make the STRIP principle work in practice, we design algorithm as following:



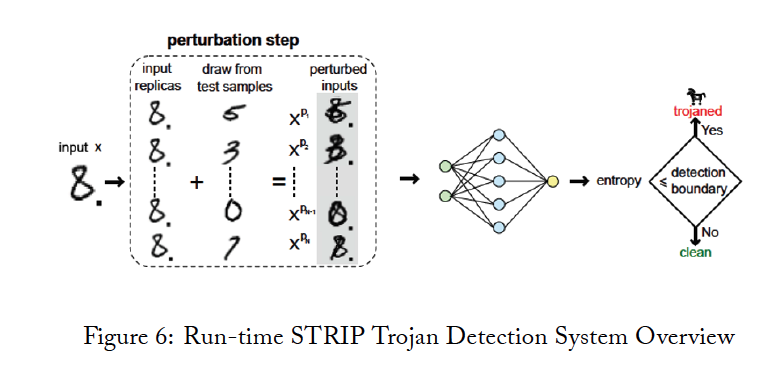
x is the input (replica), Dtest is the user held-out dataset, Fθ() is the deployed DNN model. According

to the input x, the DNN model predicts its label z. At the same time, the DNN model determines

whether the input x is trojaned or not based on the observation on predicted classes to all N perturbed inputs

{xp1 , . . . , xpN } that forms a perturbation set Dp. The judgement is based on the entropy which can be used

to measure the randomness of the prediction. Fig. 6 illustrates the whole process of the STRIP algorithm.

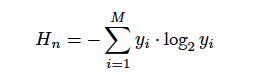


Entropy

We consider Shannon entropy to express the randomness of the predicted classes of all perturbed inputs

{xp1 , . . . , xpN } corresponding to a given incoming input x. Starting from the nth perturbed input xpn ∈

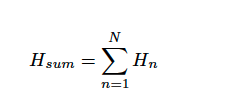
{xp1 , . . . , xpN }, its entropy Hn can be expressed:



With yi being the probability of the perturbed input belonging to class i. M is the total number of classes.

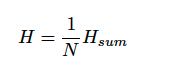
Based on the entropy Hn of each perturbed input xpn, the entropy summation of all N perturbed inputs

{xp1 , . . . , xpN } is:



With Hsum standing for the chance the input x being trojaned. Higher the Hsum, lower the probability the

input x being a trojaned input. We further normalize the entropy Hsum that is written as:



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.

7