ML for Cyber Security (ECE-GY 9163) Project Report

Kunwar Shivam Srivastav, Sagar Patel, Tamoghna Chakraborty, Sahil Makwane

kss519@nyu.edu, sp5894@nyu.edu, tc3142@nyu.edu, sm9127@nyu.edu

Environment Setup

The readme file on our GitHub repository provides a step-by-step overview of how to run our code. It also describes the dataset used and the dependencies.

Model Structure

We have designed a backdoor detector for BadNets trained on the Youtube Face Dataset. 1 shows the general model structure for all our models

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 55, 47, 3)	0	
conv_1 (Conv2D)	(None, 52, 44, 20)	980	input[0][0]
pool_1 (MaxPooling2D)	(None, 26, 22, 20)	0	conv_1[0][0]
conv_2 (Conv2D)	(None, 24, 20, 40)	7240	pool_1[0][0]
pool_2 (MaxPooling2D)	(None, 12, 10, 40)	0	conv_2[0][0]
conv_3 (Conv2D)	(None, 10, 8, 60)	21660	pool_2[0][0]
pool_3 (MaxPooling2D)	(None, 5, 4, 60)	0	conv_3[0][0]
conv_4 (Conv2D)	(None, 4, 3, 80)	19280	pool_3[0][0]
flatten_1 (Flatten)	(None, 1200)	0	pool_3[0][0]
flatten_2 (Flatten)	(None, 960)	0	conv_4[0][0]
fc_1 (Dense)	(None, 160)	192160	flatten_1[0][0]
fc_2 (Dense)	(None, 160)	153760	flatten_2[0][0]
add_1 (Add)	(None, 160)	0	fc_1[0][0] fc_2[0][0]
activation_1 (Activation)	(None, 160)	0	add_1[0][0]
output (Dense)	(None, 1283)	206563	activation_1[0][0]

Total params: 601,643
Trainable params: 601,643
Non-trainable params: 0

Figure 1: Model Structure

STRIP

Introduction

STRIP, which stands for Strong Intentional Perturbation, is a run-time based trojan attack detection system. It is used to differentiate poisoned input by trojan attacks from clean ones. Our implementation of this technique is based on the work of Gao et al. [Gao+19] .

Method

The algorithm behind the practical implementation of STRIP is as follows:

Algorithm 1 Detecting poisoned input (trojan attack) during runtime of the deployed model

function DETECTION($x, D_{test}, F_\theta()$, detection boundary)
  `trojanedFlag` $\leftarrow$ No
  for $n = 0$ to N do
    draw the n_{th} image at random, x_n^t, from D_{test}
    superimpose incoming image x with x_n^t to produce the n_{th} perturbed images x^{p_n}.
  end for
  $H \leftarrow F_\theta(D_p)$
  if $H \leq$ detection boundary then
    `trojanedFlag` $\leftarrow$ Yes
  end if
  return `trojanedFlag`
end function

Here, x is the replica of input, D_{test} is the user held-out dataset $F_\theta()$ is the deep neural network model.The model predicts its label z according to the input x. The model, at the same time, determines whether the input x is "trojaned" (poisoned) or not based on the observation on predicted classes to all N perturbed inputs $\{x^{p_1}, \ldots, x^{p_N}\}$ that forms a perturbation set D_p. Entropy is used to measure the randomness of the prediction and the result of STRIP is depended on it. The following figure, Fig. 2 gives us an overview of the STRIP algorithm process.

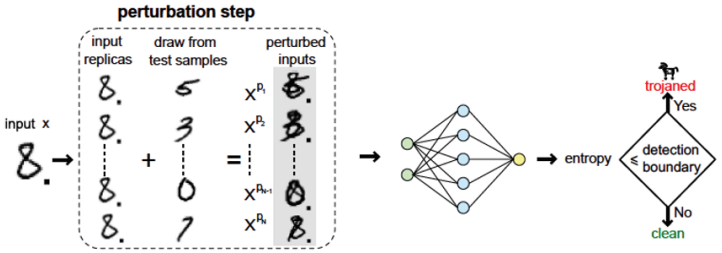

Figure 2: Run-time STRIP Trojan Detection System Overview

Neural Cleanse

Backdoor Detection

Neural Cleanse's (by Wang et al. [Wan+19]) basic concept is that when a model is poisoned, very small modifications are enough to cause the model to misclassify the target label. In our work, we iterate through all possible labels and check which one achieves wrong results for the smallest modifications. There are 3 steps tot he process:

1. Minimal trigger search. We search for a trigger window with a fixed label. It is assumed that this label is the target label of the backdoor attack trigger. Theis trigger's performance is dependent on how small are the required modifications to lead the model to misclassify all samples from other labels into the target label.

2. Iterating through all labels. We iterate through all labels in the model, for this project, its 1283. Hence, 1283 potential triggers are created.

3. Getting to the valid trigger. In the last step, we select the valid trigger from all of the 1283 triggers. This selection depends on the number of pixels the trigger tried to influence in the models. We calculate the L1 norms of all triggers. Then, the absolute deviation between all data points and the median is calculated. We divide the absolute deviation of a data point by the median and if the value is > 2, that trigger is marked as a target trigger. The "reverse trigger" is the trigger needed to repair the BadNets, and it is the target trigger which is most effective to misclassify the model.

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`.

Repairing the BadNets

The infected model is patched by pruning the poisoned neurons in the BadNet with the "reverse trigger" in order to repair the BadNet.
Some neurons were poisoned by the target triggers in the model to make it misclassify the target label, so the output value of these neurons are to be set to 0, so that the model is not affected by the target triggers anymore.
Hence, the neurons are ranked by the differences between clean and poisoned inputs produced by the "reverse triggers". We again target the penultimate layer and prune neurons in highest rank first fashion. As the performance of the model on clean targets needs to be maintained, iterations are stopped as soon as the model stops being sensitive to poisoned inputs.
You can find the details in our `repair_model.py`.

Result

```
#!/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
```

References

[Gao+19] Yansong Gao et al. “Strip: A defence against trojan attacks on deep neural networks”. In: Proceedings of the 35th Annual Computer Security Applications Conference. 2019, pp. 113–125.

[Wan+19] Bolun Wang et al. “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks”. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE. 2019, pp. 707–723.