ECE-GY 9163 ML for Security Project Report

Siyuan Shi: N18648680 ss13376
Haotian Yi: N18800809 hy1651
Shengjia Yan: N14106629 sy2703
Yifan Li: N10020433 yl7076

December 22, 2020

*Please note that we have implement two solution: a STRIP based method and a Fine-Pruning based method.

1 A Solution based on STRIP

1.1 Main idea

STRIP is a runtime detection method and its idea is based on the fact that trigger have strong effect to force result to be a fixed wrong class. For a test image, we superimpose it with random clean image for several times, if test image is poisoned, the result will be relatively static, otherwise the results will be chaos. And this is measured by entropy. Values of entropy of poison image is small and that of clean image is larger so we can compute a detection boundary.

1.2 Implementation

1.2.1 Main Functions & how they work

1. **superImpose**(overlay_img, origin_img, overlay_weight, back_weight):
Used to superimpose two images by weights, we use 0.5 and 0.9 as overlay_weight, back_weight so trigger won't be weaken.

2. **entropyCal**(background, clean_set, model, overlay_weight=0.5, back_weight=0.9):
Used to calculate mean entropy of 'background' image superimposed with randomly chosen image in 'clean_set'.

3. **getEntropyList**(x_test, x_valid, model, overlay_weight=0.5, back_weight=0.9):
Used to compute entropy lists of 'x_test' superimposed with random images in 'x_valid'.

4. **computeThreshold**(entropy_benigh, frr=0.07):
Used to compute threshold (detection boundary) between clean and poison image. This is based on the assumption that entropy list is of normal distribution. We fit entropy list into a normal distribution and we assume False Reject Rate to be 7%, we set the threshold as the value at 7% of this normal distribution.

1.2.2 Detection Procedure

1. First we use **getEntropyList** (it will call **entropyCal**) to get a list of entropy for each test image. For each test image, we will superimpose it N times with randomly chosen clean image from validation set. In our implementation, N is set to 10 according to experience and referred material.
2. Then use **computeThreshold** to compute a detection boundary between entropy distribution of poison and clean image.
3. Use **entropyCal** (it will call **superImpose**) to calculate entropy for each test image, if the entropy is less than detection boundary, the test image is judged as backdoored image, otherwise it will be judged as clean, thus we can modify output to N+1 class.

1.3 Run code

Our code is initially conducted on Colab, but we have encapsulate the code into .py program. **Please run code according to readme.md.**

*Please note that STRIP is a runtime detection method, so we have test it with sunglasses-triggered model but not test with anonymous model, because we do not have test data with anonymous trigger.

2 A Solution based on Fine-Pruning

2.1 Main idea

We implement fine-pruning defense from paper *Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks.* Fine-pruning defense is a combination of pruning and fine-tuning, and shows that it successfully weakens or even eliminates the backdoors.

We are able to disable a backdoor by removing neurons that are dormant for clean inputs. We refer to this strategy as the pruning defense. The pruning defense works as follows:

- the defender exercises the DNN which received from the attacker with clean inputs from the validation dataset, D_{valid} , and records the average activation of each channel of neurons in the final convolutional layer, which is conv_3 layer.
- The defender then iteratively prunes neurons from the DNN's final convolutional layer (conv_3 layer in BadNet) in increasing order of average activations and records the accuracy of the pruned network in each iteration.
- The defense terminates when the accuracy on the validation dataset drops below a pre-determined threshold. Here, we set this threshold to 0.94.

The average activation of neurons means the output activation value of each channel of enurons. When we prune neurons, we prune a channel of neurons at a time.

Why we prunes neurons from conv_3 layer in increasing order of average activations of each channel of neurons? Since the activation value of neuron represents the vulnerability of neuron. The smaller average activation value is, the more vulnerable the neuron is. So we start pruning from the neurons with the smallest average activation.

After we successfully prune the network, the next thing to do is fine-tuning. Instead of training the DNN from scratch, we can instead fine-tune the DNN trained by the attacker using clean inputs. Fine-tuning is a strategy originally proposed in the context of transfer learning, wherein a user wants to adapt a DNN trained for a certain task to perform another related task. Fine-tuning is significantly faster than training a network from scratch; Also, Fine-tuning using clean inputs causes the weights of neurons involved in backdoor behaviour to be updated, so that the new network won't have backdoor behavior.

2.2 Implementation

1. **superImpose**(overlay_img, origin_img, overlay_weight, back_weight):

Used to superimpose two different images. Additionally, we assign 0.5 and 0.9 as overlay_weight and back_weight. After testing numbers combination, they result a better performance.

2. **repairnet_predict**(test_input, valid_set, repair_model):

Used to calculate if the given test_input belongs to N+1 class. We randomly select ten images from validation data set. Then, we record ten different predicted classes resulting from the prediction that our repairedmodel makes given the overlay image created by test_input and one random image from validation data. Once we have the ten predicted classes in hand, we calculate the variance of them. if this result is greater than the assigned threshold value of 0.1, the function returns the predicted label. Otherwise, the function returns N+1, which has been backdoored.

2.2.1 Detection Procedure

1. To start with pruning part, we initially calculate the mean of activation at the layer "conv_3" and sort them in ascending order. We are pruning the badnet model by keep replacing the smallest average activation value with a bias penalty of value -9999 until the accuracy of the model drops below 94%.
2. We use **fine_prune** function to reduce the attack success rate of badnet.
3. Use **repairnet_predict** function to filter the backdoored images to N+1 class.

2.3 Run code

Our code is initially conducted on Colab, here is the colab link. Also we have encapsulate the code into .py program.

Please run code according to README.md.

To evaluate the backdoored model, execute fine_pruning_eval.py by running:

python3 fine_pruning_eval.py [clean validation data directory] [test data directory] [test image id] [badnet model filename] [badnet weights filename]

If the input is clean, evaluation script will return correct class, which is in [1, N]; If the input is backdoored, it will output N+1.