CS4035 - Cyber Data Analytics (Lab 4 - Group33)

Student 1 - Shipra Sharma (ID : 5093406)

Student 2 - Sudharshan Swaminathan (ID : 5148340)

We uploaded the malware_challenge folder to drive and mount the drive here. We then accessed it using this drive path, which is “/content/drive/My Drive/Lab4/malware_challenge”.

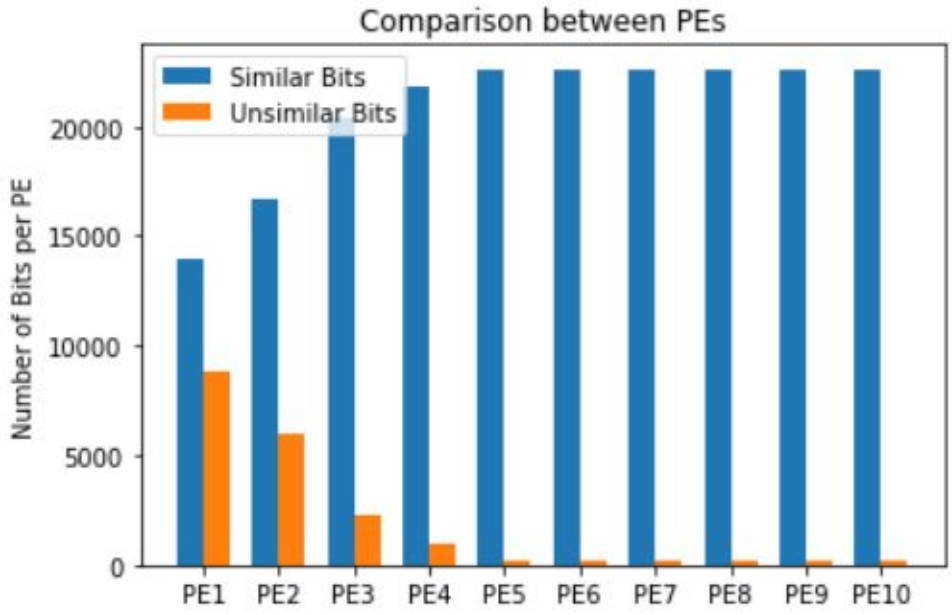
The data is uploaded separately in the runtime of the colab notebook and is extracted below. The path for this data is changed accordingly in “parameters.ini”. Here, we have the path for the data as “/content/data/”.

Task 1 - Familiarization

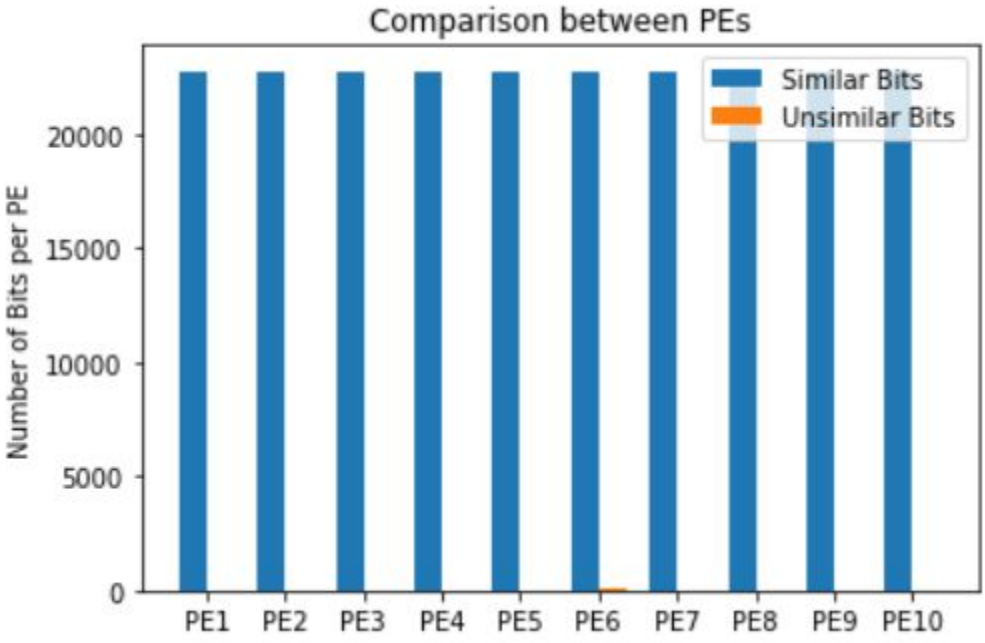
Note: To execute this task, please run **Lab4Group33Visualization.ipynb** file, because due to the limited RAM, it couldn’t be run successfully on colab environment. Also upload the attached six files: `benign`, `benign_epochs`, `malicious-before`, `malicious-before_epochs`, `adverserial` and `adverserial_epoch` in the same run environment.

In order to complete the task, as required, first we edited the parameters.ini and framework.py inorder to run the dFGSM attack. We did so using 1000 files to save the computational resources and time. The attack created a batch of malicious data points and we converted them into a file of strings for all the three types of data i.e. benign, malicious-before and adversarial data. In order to get the files in readable format, we converted the same into pandas dataframes and computed the similarities between the corresponding portable executables(PEs) in different situations.

Each data-frame row has 22761 columns which means that each PE entry has corresponding 22761 bit entries. Due to this large size we decided to visualise the comparison with the first 10 PEs of each scenario. It can be observed that without epochs, the adversary is able to change a lot of bits by comparing the malicious-before and adversarial data. Below we can see that after we train the model with 50 epochs events, the similarity between malicious files before and after adversarial attacks, increases significantly.



Malicious-Before VS Adversarial Data before epochs events



Malicious-Before VS Adversarial Data after epochs events

Thus, figure 2 clearly illustrates that the more we train models using epochs, the better hindrances we create for the adversaries to generate fake system calls. The table below illustrates the difference between malicious-before and adversarial data before and after epochs:

PE Number	Unsimilar Bits before epochs	Unsimilar Bits after epochs
PE1	8856	0
PE2	6019	0
PE3	2324	20
PE4	962	0
PE5	144	0
PE6	210	38
PE7	223	0
PE8	173	0
PE9	154	0
PE10	177	17

Task 2 - Inner maximizer

Implementing "topk" method - The "topk" function is defined in the file "inner_maximizers.py" and is based on the topk variant (Algorithm 1) as given in the GRAMS paper. As given in the algorithm, the number of bit flips is decided by the loss; greater the loss, more preferable is the number of flips for that batch. Accordingly, the batches are modified and returned. We run the framework.py using the parameters as given in the comments section in the ipynb notebook. As it can be seen that when we use topk for both training and evasion, we get an accuracy of 76.50% wherein 153/200 malwares are properly classified. Although this does not seem better than {`training_method = dfgsm_k, evasion_method = dfgsm_k`}, we observe that it does perform better and is able to evade more when dfgsm_k is used as the training method and topk as the evasion method. It is also observed that topk seems to perform better than dfgsm_k in terms of detecting malwares. topk has an accuracy of 90.5% detecting 181/200 malwares as compared to dfgsm_k's 65% which detects 130/200 malware PEs.

As for the runtime, topk has a much smaller runtime of 2-3 seconds per a batch of 10, as compared to 12-13 seconds per batch of 10 for dfgsm_k. This is observed in the output given in the ipynb file, when defending using topk and dfgsm_k as the training methods.

Comparing with baseline method (dFGSM) - On testing the implemented method topk as the evasion method with dfgsm_k being the baseline for training, it can be observed below that the accuracy of the test is 55.50%, that is, 111/200 malwares are classified properly. This is less than evading with dfgsm_k which classifies 122/200 during evasion having an accuracy of 61%. This goes to show that the evasion method using the implemented topk performs better than dfgsm_k to avoid being detected by the learned model.

Creating Adversarial Examples - Next, we train the model with topk with 3805 files (as there are 3805 malware files in the attack directory). We then use the model weights to attack and create the adversarial examples which are then stored in “aes.npy” that can be found in the submissions.