HW3 Report
Berke Can Rizai 69282

Part - 1

a) In the eqn. (1), we have the following:

$$\mathrm{P}_S(w) \quad = \quad \frac{N_H N_S(w)}{N_H N_S(w) + N_S N_H(w)} \qquad (1)$$

Where Nh, Ns, Ns(w) and Nh(w) are counts calculated for each token w. In the Bayesian method, the learning algorithm is naive which means different tokens are not interacting, which would eliminate the assumption in the 3.4. These scores are then added to the prior belief, which also doesn't break the assumption. Then, exponential smoothing is applied.
Later, scores that are calculated are still not letting scores of sparse tokens interact.

$$I(E) \quad = \quad \frac{1 + H(E) - S(E)}{2} \in [0, 1] \ , \qquad (3)$$

$$H(E) \quad = \quad 1 - \chi^2_{2n} \left(-2 \sum_{w \in \delta(E)} \log f(w) \right) \ , \ (4)$$

Something like attention weights for example would let the model learn from token interactions.

b) The observation that "the spam scores of distinct words do not interact" is used to guide the attack. When trying to increase the expected spam score of target email.
Since adding a word to the attack email will not change the scores of other words, optimal strategy in paper is to include as many words as possible while creating email.
"I is monotonically non-decreasing in each f(w)" is another observation. This means we can add many words, however it shouldn't be infinitely large.
This will maximize the expected spam score of the target email, since the spam score of the target email is simply the sum of the final scores for each word it contains.

Researchers add that, if the attacker has specific knowledge domain or background knowledge of other sort etc., the optimal attack payload will include all words in the target email.

c) In this technique, observation is that attacks increase both ham and spam scores of emails. Proposed θ0, θ1 are used to classify even after attacks. These parameters are dynamically adjusted, and they overcome attacks that shift scores of all samples.

After splitting mails to filter and validation sets, they used the g function to select a threshold and the aim of this function is to in a sense, normalize the classification amount. Function g is following, g(t) = NS,<(t) (NS,<(t) + NH,>(t))−1 . Here, variables are explained as such from the text, “NS,<(t) is the number of spam emails with scores less than t and NH,>(t) “.

d) Some features to be used in outlier detection would be things such as number of different words in mail, number of total words, number of times words are used ie. distribution of pdf of words would be meaningful. Some other feature could be the ordering of words, since in the attack ordering could be off, this would easily tell that the email is outlier as well. We could use isolation forest or one class SVM for this.

e)

$$\underset{s \in \mathcal{S}}{\text{minimize}} \;\; F(s) = f(\boldsymbol{x} \oplus s) + \lambda \cdot \mathcal{C}(s),$$
$$\text{subject to } q \leq T.$$

In this optimization problem,
x is denoting malicious input program that is in terms of bytes
s are the different manipulations that will be applied to x and there k many manipulations s.
$\oplus$ is a function that denotes that manipulation s is applied to program x and then returns the malicious applied program and preserves functionality while doing this.
f is denoting classifier, where it takes the program as input and returns score of probability of the input being malicious. If it is lower, program is classified as benign.
C is number of infected bytes to be inserted into program that is formulated like a penalty function on s
F, objective function is sum of classification output from the f and scaled penalty from the C. lambda scale is the scaling factor of malicious input size that lets us evaluate the effect of increased size of attack. This objective is minimized however one thing to note here is that if we increase the s, C will increase but f will decrease and vice versa which is the conflicting part.
T is the query budget denoting maximum number of times we can query the model and q is the query.

f) Functionality preserving is denoting that after injecting the malicious bytes to the program, regardless of where we inject or how much we inject, program will behave exactly the same from the functionality perspective. It won’t stop performing its tasks or behave differently compared to before attack was done. It is functionality preserving since malicious bytes are appended at the end of a file or in newly created parts and injected bytes are benign on their own, they don’t change the behavior of the program. Researchers classify three types of changes that can be done for preserving functionality, first is structural modification, here behaviour stays exactly the same and

only changes are made in file. Some examples are padding, filling empty space, injecting imports etc.
Second is behavior modification, where behaviour is changed, but the main function of the program is preserved. These including modifying specific parts, calling some other functions that turn back to main program after execution or binary injection etc.
Last is padding and section injection via GAMMA.

For the query efficient, from the paper, it is noted that other attacks are “they are query-inefficient, as they rely on iteratively applying random transformations to the input malware”. In the proposal of paper, attacks are extracted from benign samples and not random which makes it more efficient to get some meaningful results rather than trying many things that would evaluate to basically nothing. With the GAMMA model, number of queries are important since they will be deciding factor as we only have black-box access to the model

g) Here, these results are intuitive because the GAMMA model can learn more if the budget for number of queries is higher. If it can query the model say, 200 times, it can only update its parameters (injections) that many times (200) and model might not converge to optima in that case. However, if we allow it to query more times, it will learn better to update the outputs so that it will have higher success rate and less detection rate. It will learn to create attacks that are stealthier and more evasive. It is quite similar to optimization in deep learning, where convergence to optima takes epochs to complete.
Detection rate is how much the attacks are detected by the virus detection ML models, MalConv and GBDT. It is calculated with 500 random samples from the dataset of 15000 inputs of malware and 15000 inputs of benign data.
Attack size is the number of bytes to be injected into program or appended at the end.
Number of queries is the number of times model was evaluated by ML system.
As I have discussed, as number of queries grow, detection rate will decrease since GAMMA will be better. We can see that in the padding attacks in figure 4, after some point of attack size, model performance starts degrading and detection rate starts rising. But both in padding and injection attacks, small values of attack size has higher detection rate. In the injection attack, attack after some point of attack size, performance (detection rate) doesn’t change.

h) Researchers observed that model is transferable. They have tried over 12 antivirus models to test it. In the following table below, we can see that model has similar performance for some of the antivirus programs but not all. It should be also noted that these antiviruses that were evaded by the proposed GAMMA all has some static feature extraction which could be the reason for their vulnerability and thus the transferability of attack.
As discussed in the paper, different implementations and approaches by the anti virus, drastically effect the performance in transfer.

	Malware	Random	Sect. Injection
Detections (VT)	46.56 ± 12.40	40.80 ± 12.40	34.50 ± 12.63

TABLE IV

DETECTION RATE OF 9 ANTIVIRUS PROGRAMS FROM VIRUSTOTAL COMPUTED ON (i) THE INITIAL SET OF 200 MALWARE SAMPLES, AND ON THE SAME SAMPLES MANIPULATED WITH (ii) RANDOM ATTACKS AND (iii) SECTION-INJECTION ATTACKS

	Malware	Random	Sect. Injection
AV1	93.5%	85.5%	30.5%
AV2	85.0%	78.0%	68.0%
AV3	85.0%	46.0%	43.5%
AV4	84.0%	83.5%	63.0%
AV5	83.5%	79.0%	73.0%
AV6	83.5%	82.5%	69.5%
AV7	83.5%	54.5%	52.5%
AV8	76.5%	71.5%	60.5%
AV9	67.0%	54.5%	16.5%

Also in the table III,

TABLE III

NUMBER OF ANTIVIRUS PROGRAMS FROM VIRUSTOTAL (VT) THAT DETECT (i) THE INITIAL MALWARE AND ITS MODIFIED VERSIONS WITH (ii) RANDOM AND (iii) SECTION-INJECTION ATTACKS, AVERAGED OVER 200 MALWARE SAMPLES (STANDARD DEVIATION IS ALSO REPORTED). WHILE RANDOM ATTACKS EVADE 5.76 DETECTORS, ON AVERAGE, SECTION-INJECTION ATTACKS EVADE UP TO 12.01 DETECTORS

	Malware	Random	Sect. Injection
Detections (VT)	46.56 ± 12.40	40.80 ± 12.40	34.50 ± 12.63

We can see that Section injection attacks could evade 12 detectors at average. And authors add that if the adversary model was further optimized, transferability would be easier to observe.

2-

Q1) Here, I have run experiments where I flip labels randomly for 100 times and took the mean for each run.

Results are:

```
Accuracy of poisoned DT 0.05 : 0.9604081632653062
Accuracy of poisoned DT 0.1 : 0.9324489795918366
Accuracy of poisoned DT 0.2 : 0.8889795918367345
Accuracy of poisoned DT 0.4 : 0.6728571428571428
Accuracy of poisoned LR 0.05 : 0.966326530612245
Accuracy of poisoned LR 0.1 : 0.9424489795918365
Accuracy of poisoned LR 0.2 : 0.9018367346938776
Accuracy of poisoned LR 0.4 : 0.743877551020408
Accuracy of poisoned SVC 0.05 : 0.9926530612244897
Accuracy of poisoned SVC 0.1 : 0.9879591836734692
Accuracy of poisoned SVC 0.2 : 0.9726530612244896
Accuracy of poisoned SVC 0.4 : 0.7777551020408161
```

Q2)

```
Recall of inference attack 0.99 : 0.43
Recall of inference attack 0.98 : 0.52
Recall of inference attack 0.96 : 0.59
Recall of inference attack 0.8 : 0.81
Recall of inference attack 0.7 : 0.93
Recall of inference attack 0.5 : 1.0
```

Here, lower value of t such as 0.5 gave better recall and it means that whenever probability was higher than 0.5 model was sure that its from training data and it was right. Whenever t is lower, more data is regarded as positive and recall is going higher. This could be different if there was more than 2 classes like 8 or 9 etc.

Q3)

```
Success rate of backdoor: 0.0 model_type: DT num_samples: 0
Success rate of backdoor: 0.5102040816326531 model_type: DT num_samples: 1
Success rate of backdoor: 1.0 model_type: DT num_samples: 3
Success rate of backdoor: 1.0 model_type: DT num_samples: 5
Success rate of backdoor: 1.0 model_type: DT num_samples: 10
Success rate of backdoor: 0.0 model_type: LR num_samples: 0
Success rate of backdoor: 0.46938775510204084 model_type: LR num_samples: 1
Success rate of backdoor: 1.0 model_type: LR num_samples: 3
Success rate of backdoor: 1.0 model_type: LR num_samples: 5
Success rate of backdoor: 1.0 model_type: LR num_samples: 10
Success rate of backdoor: 0.0 model_type: SVC num_samples: 0
Success rate of backdoor: 0.0 model_type: SVC num_samples: 1
Success rate of backdoor: 0.0 model_type: SVC num_samples: 3
Success rate of backdoor: 0.0 model_type: SVC num_samples: 5
Success rate of backdoor: 0.0 model_type: SVC num_samples: 10
```

My trigger patterns are:
For SVC, I flipped the support vector decisions since they can make the most impact and model is built by them in literal sense.

For the decision tree, I randomly generated data that is from the same distribution of the training data and made the dt classify them. With the predictions coming from the dt, I

flipped labels and added them to the training data. If I slightly modified the data X_train and gave it to DT, chances are nothing would change since it could be the case that DT could just set threshold near them and have same classifications.

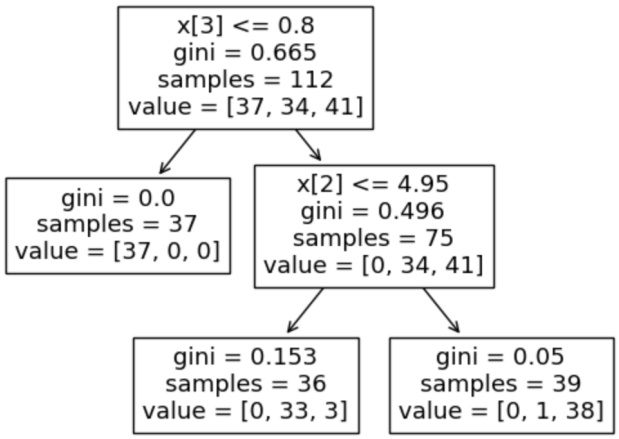
For the Logistic regression, I took the most important coefficient and took data that is sampled with label 1 and randomly sampled some more data and flipped labels which would render the data of LR that it relied on as feature useless since now it gave loss if it tried to decide with that feature.

My success rate is the percentage of predictions that are flipped compared to the non poisoned data. If all predictions are same, it had success of 0 however, it all predictions changed, it was successful attack. Percantage of labels flipped is ratio I defined.

1 * (pred_orig != pred_new) / count(pred_orig) where left side are counts of the changed.

Q4)
My strategies for each of the model types are different, in the decision tree, I follow the decision graph and in order to change the label, I made it so that corresponding value in the modified example is just a little above or below the threshold thats determined by the tree.

I took a look at some examples from the, sklearns official website.
https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html



For example, here we see an example tree. I followed the tree and changed values on the example.

In the logistic regression, I look for the most important features and changed each of them a bit, if decision shifted towards the other side (probability decreased) I continued

shifting that feature value to that side and else, I changed the direction (ie started increasing that feature) until the predicted label has changed.

In the SVM, since labels are flipped at the support vectors that are creating SVC, I looked at the vectors that would flip decision and are closest to the example, and basically shifted my example to that vector. This guarantees that label will be flipped.

Perturbation table:

```
Avg perturbation for evasion attack using DT : 1.1644999041461266
Avg perturbation for evasion attack using LR : 3.5141955961253046
Avg perturbation for evasion attack using SVC : 5.29675
```

(Right bottom of the image is not dirt on screen, its text but cropped :D)

Q5)

```
Here, you need to conduct some experiments related to transferability and prin
their results...
Transferabilities
DT - LR 0.025000000000000022
DT - SVC 0.9
LR - DT 0.025000000000000022
LR - SVC 0.9
SVC - DT 0.0
SVC - LR 0.375
```

Here, I have printed as probability of transfer where higher value means more transferability. I basically checked before and after perturbation results if they are same and if so, I added 1 to list else 0. Then took mean and deducted from 0 to show higher score if we see more transferability.

```
DT - LR 39
DT - SVC 4
LR - DT 39
LR - SVC 4
SVC - DT 40
SVC - LR 25
```

And these are a number of same decisions and if we deduct this from 40, we can see how many could transfer.

We can see that data perturbed from other models are quite transferable to SVC however, data that was evaded from SVC classifier were not transferable to other models, this could be because of the nature of SVC where support vectors sometimes doesn't make much sense.

Q6)

Here, I queried the original model and after querying it with number of data given.

```
******************************
Number of queries used in model stealing attack: 8
Accuracy of stolen DT: 0.9591836734693877
Accuracy of stolen LR: 0.9183673469387755
Accuracy of stolen SVC: 0.6938775510204082
******************************
Number of queries used in model stealing attack: 12
Accuracy of stolen DT: 0.9795918367346939
Accuracy of stolen LR: 0.8775510204081632
Accuracy of stolen SVC: 0.7755102040816326
******************************
Number of queries used in model stealing attack: 16
Accuracy of stolen DT: 0.9795918367346939
Accuracy of stolen LR: 0.8979591836734694
Accuracy of stolen SVC: 0.9183673469387755
******************************
Number of queries used in model stealing attack: 20
Accuracy of stolen DT: 0.9795918367346939
Accuracy of stolen LR: 0.9795918367346939
Accuracy of stolen SVC: 0.9591836734693877
******************************
Number of queries used in model stealing attack: 24
Accuracy of stolen DT: 0.9795918367346939
Accuracy of stolen LR: 1.0
Accuracy of stolen SVC: 0.9795918367346939
```

We can see that we can steal logistic regression and decision tree models with minimum number of queries while we need a bit more data for SVC classification model.