COMP 530 Data Privacy and Security Homework 3 Report

Due December 27th

Ismayil Ismayilov

Part 1: Reading

A. Is the paper's attack a training-time attack or test-time attack? Explain briefly.

The attack described in the paper is a test-time attack. The paper mentions multiple times that the proposed attack is a black-box attack that has only query access to the target model which makes it a test-time attack.

B. What kind of analysis does the attack mainly rely on: static analysis or dynamic analysis? Explain briefly

The attack targets machine learning based malware detectors that rely on static analysis. This is mentioned in the paper as

> We focus on two popular learning-based Windows malware detectors, built on features extracted from static code analysis

When discussing the limitations of their work, the authors mention that their approach is not effective against detectors which utilize dynamic analysis.

C. What is meant by *query-efficiency* and *functionality-preserving* in the context of this paper? Why are these two properties important/desired?

In the context of the paper, query-efficiency means that the number of queries an attacker needs to pose to the target model in order to bypass the detector is comratively low. Why this would be desirable is clear since, by definition, a query-efficient attack would take less time to bypass the detector. In a real-life environment, it could also perhaps lead to less suspicious behavior and lower detection rates (if the detector being attacked is remote, an attack that is not query-efficient would likely send many queries and arouse suspicion)

Functionality-preserving in the paper is used to refer to manipulations that can be applied to the input malware program without compromising its integrity. The paper mentions that they achieve this property by injecting content specifically targeted to achieve evasion (content extracted from benign samples). The reason why functionality-preserving transformations would be desirable is that they end up being more efficient; attacks which do not apply-functionality preserving transformations "may require executing the adversarial malware in a sandbox at each iteration of the optimization process, to ensure that its intrusive functionality is preserved."

D. The paper discusses two existing methods: MalConv and GBDT. One of them is more similar to the *extract relevant features, then train ML model* pipeline we learned in class, whereas the other one is different in that the feature extraction step is omitted. Which one is which, and why?

GDBT is the method which uses the traditional ML pipeline of extracting features and applying an ML model (Gradient Boosting Decision Trees, in this case). This is stated in the paper as

> Differently from MalConv, this detector uses a fixed representation consisting of 2,381 features, extracted from ...

The MalConv model uses convolutional neural networks and differs significantly from the GDBT model. In essence though, the CNN based approach could also be considered an instance of the *extract relevant features, then train ML model* paradigm. The CNNs are used to extract the relevant features and then the ML model is trained on them. However, the GDBT model is closer to the traditional definition of the paradigm.

E. What do each of the following terms/notation in the attack formulation stand for?

- s: A vector of k elements parametrized in $[0, 1]$. Each element of s corresponds to a different manipulation that can be applied to the malicious program; the element itself denotes the degree to which the manipulation is applied. A manipulation is defined as the injection of content extracted from a predefined set of k benign sections.
- S: Denotes the sample space of all possible transformation vectors
- $F(s)$: Denotes the objective over which the minimization is performed. Combines two conflicting terms $f(x \oplus s)$ and $C(s)$ which will be explained later
- x: Denotes the malicious input program, described a string of bytes of arbitrary length
- $\oplus : X \times S \to X$: A function that applies the manipulations as described by transformation vector s to the input program x and returns the manipulated program. The applied manipulations preserve the program's functionality.
- λ: A hyperparamater that tunes the trade-off between the terms $\oplus : X \times S \to X$ and $C(s)$. Promotes solutions with smaller number of injected at the expense of reducing the probability that the same sample is misclassified as benign.
- $C(s)$: A penalty function that evaluates the number of injected bytes into the input malware.
- q: Denotes the number of queries that can be made.
- T: A hyperparameter that serves as an upper bound on the maximum number of queries q that can be made. Increasing the query budget T allows the attack to better optimize the trade-off between misclassification confidence and payload size at the expense of an increased computational complexity.

Explanation: After the transformation is applied to the input program, the objective is trying to minimize the probability that the new input is classified as malicious while making sure the transformation does not inject too many bytes into the program. This minimization is performed over all valid transformations. Additionally, the number of queries that can be made to the target model is constrained by the query budget.

F. According to the results in Figure 4, under the same attack size, higher number of queries decreases detection rate. Why is this an intuitive result? Explain by discussing: (i) what detection rate, attack size, number of queries mean, and (ii) how you would expect attack size and number of queries to affect detection rate in the context of this paper's attack

In the paper, the detection rate refers to the ratio of malicious samples that were classified as malicious to the total number of samples. The attack size refers the number of bytes that were injected into the input malware. The number of queries refers to how many times the target model was queried or, in other words, how many queries were asked to the model to find out whether the given input program is malicious or benign.

It is expected that as the number of queries increases, both the detection rate and the attack size would decrease. Increasing the number of queries allows the attack to better optimize the trade-off between misclassification confidence $f(x \oplus s)$ and the attack size $C(s)$. Essentially, increasing the number of queries allows the optimizer to find the more effective solutions that would not be found early in the search process. This intuition is confirmed by the following paragraph from the paper

> Another significant effect is posed by the number of total queries used by the genetic optimizer: the more are sent, the better the adversarial examples are in both detection rate and size. Intuitively, by sending more queries, GAMMA can explore more solutions that are stealthy and evasive at the same time, but such solutions could not be found at early stages of the optimization process.

G. Do the attacks on commercial antivirus software leverage the concept of transferability? Why or why not?

The attacks likely do leverage the concept of transferability. This is confirmed by the reasoning provided in the paper

> The reason may be that some of these antivirus programs already use static machine learning-based detectors to implement a first line of defense when protecting end-point clients from malware, as also confirmed in their blog or website, and this makes them more vulnerable to our attacks.

Assuming that it is true that the bypassed detectors are based on machine learning (which seems likely given the above), the fact that they are bypassed would indicate transferability. This is because, as per the definition of transferability, an attack designed against one model (GDBT) is successful against another model.

Part 2: Implementation

Label Flipping Attack

	Model Type		
n	DT	LR	SVC
0.05	0.5602	0.6564	0.5923
0.10	0.5651	0.5437	0.5644
0.20	0.5091	0.5863	0.4985
0.40	0.5092	0.4906	0.4943

Table 1: Accuracies of poisoned models

The accuracies of poisoned models depending on the what percentage of the data was poisoned are shown in the table above. The accuracies of the original models are 0.9806, 0.9854, 0.9684 for DT, LR and SVC respectively.

In all cases, poisoning even 5% of the data leads to a sharp drop in accuracy. In the case of DT, the accuracy drop is 42% compared to the original model. In this case, LR is best performing one with an accuracy of 65.64%.

As expected, the accuracy keeps dropping as n is increased. Perhaps surprisingly, however, some models experience a brief increase in accuracy as n is increased. This can be observed in the case of DT as the accuracy increases from 56.02% to 56.51% as n goes from 0.05 to 0.10. The same effect can be seen when the accuracy of LR increases from 54.37% to 58.63% as n goes from 0.10 to 0.20.

As was mentioned previously, LR is the best performing model for $n = 0.05$ with an accuracy 65.64%. In this context, LR can be said to be the most robust model. However, both LR and SVC experience sharp drops in accuracy as n eventually increases to 0.40. The accuracy drop of DT from $n = 0.05$ to $n = 0.40$ is comparatively low (roughly 5%) which would make it the most robust of the three models.

Backdoor Attack

	Model Type		
Number of samples	DT	LR	SVC
0	0.0	0.0	0.0
1	0.163	0.0	0.198
3	0.826	0.817	0.742
5	1.0	0.897	0.996
10	1.0	1.0	1.0

Table 2: Success rates of backdoor attacks

The success rates of the backdoor attack depending on the number of samples backdoored samples and model type are summarized in the table above. Additionally, the constants used in the attack are defined below:

- `BACKDOOR_TRIGGER_FLAG` - A flag value that some features of a backdoored sampled are set to. In the current implementation, the value is 1000
- `BACKDOOR_NUM_FLAGGED_FEATURES` - Defines how many features starting from 0 will be set to a value of `BACKDOOR_TRIGGER_FLAG`. In the current implementation, the value is 2 meaning that the first two features will be set to `BACKDOOR_TRIGGER_FLAG`
- `BACKDOOR_RANDOM_NOISE_VARIANCE` - The value used for the variance of the normal distribution when drawing samples for features that were not flagged. In the current implementation, the value is 5.0
- `BACKDOOR_NUM_TEST_SAMPLES` - The number of generated samples when testing the accuracy of the backdoor attack. In the current implementation, the value is 1000

Trigger Pattern

The trigger pattern is defined as follows. To backdoor a sample, the first `BACKDOOR_NUM_FLAGGED_FEATURES` are set to a value of `BACKDOOR_TRIGGER_FLAG` while the others are drawn from a normal distribution with mean 0 and variance `BACKDOOR_RANDOM_NOISE_VARIANCE`. Therefore, given the values in the current implementation, the first 2 features of a sample are set to a value of 1000 while the others are drawn from a normal distribution with mean 0 and variance 5.0. Additionally, the label for the backdoored sample is set to 1. The relevant code listing summarizing the backdoor sample generation is shown below

```
1  def generate_backdoored_samples(num_samples, num_features):
2      backdoored_x_samples = []
3      backdoored_y_samples = np.ones(num_samples)
4  
5      for _ in range(num_samples):
6          sample = np.random.normal(0, BACKDOOR_RANDOM_NOISE_VARIANCE, num_features)
7          sample[:BACKDOOR_NUM_FLAGGED_FEATURES] = BACKDOOR_TRIGGER_FLAG
8  
9          backdoored_x_samples.append(sample)
10 
11     backdoored_x_samples = np.reshape(backdoored_x_samples, (-1, num_features))
12 
13     return backdoored_x_samples, backdoored_y_samples
```

In line 6, the random sample is drawn from the normal distribution as stated before. In line 7, the first two features are set to the value of the trigger flag. Note that, the setting of the label to 1 is done at Line 3 by calling *np.ones*()

Success Rate Measurement

To measure the success rate of the experiment, I generate 1000 backdoored samples as described before. The previously trained backdoored model is used to predict labels for the backdoored samples. The success rate of the experiment is defined as the proportion of backdoored samples that the backdoored model predicts to have a label of 1.

Evasion Attack

The average perturbation of the evasion attack during my run was 4.5628

Attack Strategy

In a loop, while the predicted class of the perturbed sample is the same as that of the actual sample I do the following:

- Pick a random feature
- Set the value of the feature to a random sample drawn from a normal distribution with mean `actual_feature_val` and variance `EVASION_RANDOM_NOISE_VARIANCE` where `actual_feature_val` is the feature value of the actual (unperturbed) example and `EVASION_RANDOM_NOISE_VARIANCE` is a predefined constant (1.5 in the current implementation).

The relevant code listing is shown below.

```
    while pred_class == actual_class:
        rand_idx = np.random.choice(range(num_features))
        modified_example[rand_idx] = np.random.normal(
            actual_example[rand_idx], EVASION_RANDOM_NOISE_VARIANCE
        )

        pred_class = trained_model.predict([modified_example])[0]
```

Note that, in the context of this attack strategy, the maximum perturbation is upper bounded by `num_features` multiplied by `EVASION_RANDOM_NOISE_VARIANCE` where `num_features` is the number of features. Since the current implementation has 1.5 for `EVASION_RANDOM_NOISE_VARIANCE` and 4 for `num_features`, the upper bound on the maximum perturbation will be $4 * 1.5 = 6.0$. Note that, this is the maximum upper bound and the observed perturbation is usually less than 5.

Transferability of Evasion Attacks

The evasion attack transferability experiment results are as follows:

- For DT, out of 100 samples
  - 39 transferred to LR
  - 44 transferred to SVC
- for LR, out of 100 samples
  - 45 transferred to DT
  - 63 transferred to SVC
- for SVC, out of 100 samples

  - 46 transferred to DT
  - 68 transferred to LR

Additionally I define a custom metric for evaluating transferability as the number of total successful transfers divided by the number of model pairs evaluated. In this case, there are $39+44+45+64+46+68 = 306$ and 6 model pairs, which would make the metric equal to $\frac{306}{6} = 51$. Across the runs, the value of the metrics hovers around 50 but the values may differ from run to run. Additionally, note that the range of values for the metric is $[0, 100]$. Essentially, the metric shows how many samples out of 100, transfer on average across model types.

While larger values would, of course, be more desirable, I would posit that the value of 51 for the transferability, is large enough to consider the evasion attack as highly cross-model transferable.

That being said, the implemented metric is a bit naive in that it either understates or overstates the results of transfers across a single model pairing. This can be seen by considering that the SVC-LR model pairings have consistently higher transfer rates which pushes the average upward. This can be misleading as the other model pairings SVC-DT and LR-DT have consistently lower transfer rates. Therefore, I would consider it more instructive to look at the transfer rates individually and not relying on the metric alone.

Model Stealing

	Model Type		
Number of queries	DT	LR	SVC
5	0.4927	0.8617	0.6408
10	0.7670	0.9005	0.6383
20	0.9029	0.9757	0.6432
30	0.9005	0.9660	0.6456
50	0.8981	0.9636	0.7549
100	0.9393	0.9806	0.7306
200	0.9612	0.9806	0.7816

Table 3: Accuracies of stolen models

It is clear from the table above, that LR is the easiest model to steal as the stolen model achieves a high accuracy of 86.17% even with 5 queries. LR continues being the best performing model across all number of queries with the model at number of queries of 200 achieving equal performance with the original LR model.

DT is the second easiest to steal although its performance when the number of queries is low is far worse than the performance of LR at the same number of queries. DT starts achieving high performance after the number of queries eclipses 20. The model at 200 queries is almost as accurate as the original DT model.

SVC is the hardest model to steal. Its performance hovers around low sixties for up to 30 queries. The performance significantly improves after the number of queries reaches 50 but the final performance at query number 200, while passable, is not close to the performance of the original SVC model.

List of dependencies is provided in the *requirements.txt* file