Replication Study: Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN

Claire Casalnova
Department of Computing Security
Rochester Institute of Technology
Rochester, NY 14623
ccc2876@rit.edu

1 Summary

Recently, machine learning algorithms are being implemented to detect malicious software within systems. Malware authors are learning how to attack these machine learning detection algorithms in order to create malware that is able to bypass detection by the algorithm. However, malware authors do not have access to the inner architecture of the algorithm that they are trying to attack and therefore they can only perform a black box attack. I chose to do a replication study on the paper *Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN* that seeks to create adversarial examples that are successfuly able to fool a black box machine learning malware detection system. It does this by utilizing a Generative Adversarial Network (GAN) named MalGAN to create malware samples that appear benign and then refit the black box detection system to read these samples as benign. After the black box is retrained, MalGAN is successfully able to trick the black box into believing that the malware samples are benign.

2 Background

2.1 Generative Adversarial Network

A Generative Adversarial Network (GAN) is comprised of two feed forward neural networks, a generator and a discriminator, that work in competition with each other. The generator's goal is to create adversarial examples and the discriminator's goal is to determine if the examples that it is fed are real or fake. The generator is trained by receiving the output of the discriminator as well as labels that associate the output as real. The discriminator train on fake data by taking the generator's fake data and labeling it as adversarial and trains on real data by feeding in real data with labels that associate it as real examples.

2.1.1 Generator in MalGAN

The generator is the portion of the architecture that is tasked with creating the adversarial examples. It is fed a binary vector that represents a piece of malware and its corresponding features. The examples that are created by the generator have changed bits of the features within the sample of malware making it appear as though it is a benign. The generator returns a binary vector of malware samples that appear benign to the black box.

2.1.2 Discriminator in MalGAN

The discriminator is tasked with determining whether the sample it was fed is a malicious program or a benign program. The discriminator is used to train the generator by providing gradient information. The discriminator acts similarly to the black box since the malware authors are unable to determine the specific detection algorithm.

2.2 Black Box

The black box is utilized as the classifier for malware. These are different machine learning algorithms that a machine learning malware detection system would use to determine if a sample it is sent is malware or not. For the purpose of the replication study I chose to examine three different machine learning algorithms as the black in order to compare the output as to which algorithms were more easily fooled than others.

2.2.1 Random Forest Classifier

A Random Forest Classifier (RF) is a machine learning model that is comprised of many decision trees. A random forest learns from a random sample of the data points that are passed to it. It trains each tree on the subset of these points and as the tree depth goes down the number of features within a node decreases. The random forest makes its final predictions by averaging the predictions of each of the trees.

2.2.2 Support Vector Machine

A Support Vector Machine (SVM) is a supervised learning algorithm that is utilized for binary classification problems. After training on data, the SVM is able to classify new data into one of the categories. For this replication study, I utilized a linear SVM. This algorithm learns by plotting the data points and clustering them based on their labels and drawing a line that separates the two different classes. This line is used as the decision line and wherever new data falls on the plot it is classified into that pre-determined class.

2.2.3 Logistic Regression

Logistic Regression (LR) is used as a classification algorithm for a discrete number of outputs. Logistic regression uses the sigmoid function to output a probability that can then be mapped to a binary classification.

3 Data

The first step I took in replicating this paper was understanding the data sets that were provided in the Github that was associated with the paper. Three data sets were listed *data.npz*, *data1.npz*, and *mydata.npz*. There was no real explanation on what each of these data sets contained or how they were different. By analyzing each of the data sets, I found a flaw within the *mydata.npz* file. Both the benign software and malicious software labels were set to 1 so I decided to rule out this data set for use. *data.npz* was the updated version of *mydata.npz* so I additionally ruled this set out and decided to solely use *data1.npz* as there were no flaws or errors within this set.

Within this data set, there were 1,368 samples of malware and 441 samples of benignware. Each of these samples contained 128 features that were the API calls outputted from running the samples through Cuckoo sandbox. The data was then normalized by designated an API call as either 1 or 0 indicating that a piece of software either had that API call or did not.

4 Architecture Specifications

The authors of the paper specified that their generator and discriminator were "multi-layer feed-forward neural networks." I decided to start my experiment by examining the results between two hidden layers and three hidden layers within both the generator and discriminator and see if the results varied between the two.

Additionally, I decided to test the results between using the sigmoid activation function for the output layer and the tanh function. The tanh function provides stronger gradients over the sigmoid function so I wanted to see if this would result in the adversarial examples being able to fool the black box easier than when using the sigmoid.

I tested each of these parameters on all three of the black box algorithms I chose to experiment with and the next section will outline the analysis of the results I achieved.

I chose to train MalGAN for 200 epochs before retraining the black box and then for 75 epochs after the retrain. Additionally, I chose a batch size of 64 samples when training MalGAN and chose to use the Mean Squared Error as the loss function.

5 Results

5.1 Two Hidden Layers using Sigmoid

The following tables display the final results from a two hidden layer network utilizing the sigmoid function for each of the black box algorithms

Black Box Algorithm	Test TPR Before Retraining	Test TPR After Retraining
Random Forest	0.9818	0.9547
SVM	0.9790	0.20468
Logistic Reg	0.9918	0.0175

5.2 Two Hidden Layers using Tanh

The following table displays the final results from a two hidden layer network utilizing the tanh function for each of the black box algorithms

Black Box Algorithm	Test TPR Before Retraining	Test TPR After Retraining
Random Forest	1.0	0.9825
SVM	0.9927	0.2733
Logistic Reg	0.9818	0.2325

5.3 Three Hidden Layers using Sigmoid

The following table displays the final results from a three hidden layer network utilizing the sigmoid function for each of the black box algorithms

Black Box Algorithm	Test TPR Before Retraining	Test TPR After Retraining
Random Forest	1.0	0.9927
SVM	0.9854	0.2193
Logistic Reg	1.0	0.0614

5.4 Three Hidden Layers using Tanh

The following table displays the final results from a three hidden layer network utilizing the tanh function for each of the black box algorithms

Black Box Algorithm	Test TPR Before Retraining	Test TPR After Retraining
Random Forest	1.0	0.9971
SVM	1.0	0.4576
Logistic Reg	1.0	0.4795

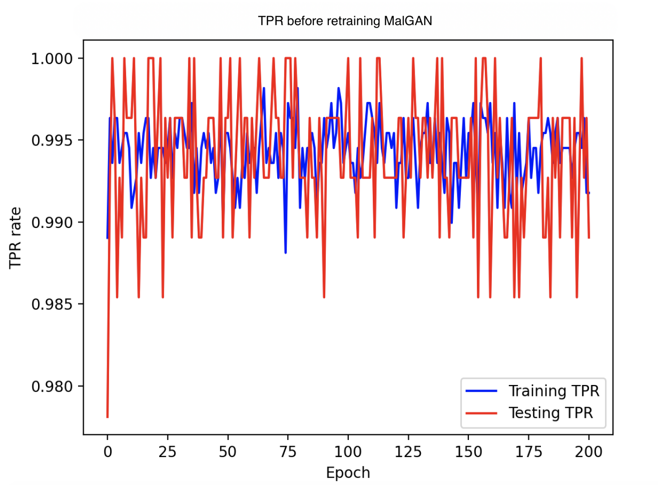
The best results from each testing setup are highlighted and the best result across all of the testing setups is highlighted in green.

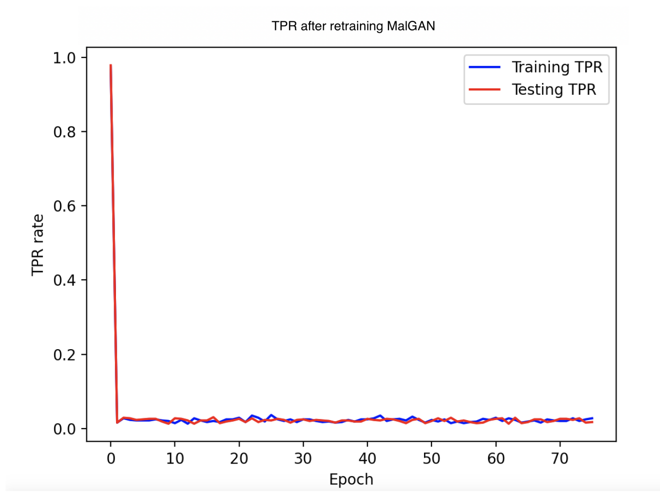
6 Analysis

Overall, I was unable to reproduce the exact results of the paper. However, I was able to reproduce results where the black box was deceived by the adversarial examples a majority of the time.

The most noticeable difference between my results and the results of the paper was in relation the the Random Forest classifier. Even after retraining it with adversarial examples, the Random Forest was not fooled by the adversarial examples and was still able to detect them over the Support Vector Machine and the Logistic Regression algorithms.

The logistic regression black box algorithm performed the best over all the other algorithms and it had the lowest true positive rate (TPR) when MalGAN had two hidden layers and used sigmoid activation. Thus it would be the best option for malware authors to use this version of MalGAN when trying to trick malware detection systems into believing that their samples are benign. The below figures outline the Logistic Regression's TPR over time before and after retraining MalGAN:





My created linear Support Vector Machine (SVM) was able to fool the black box in approximately 75% of the cases.

The best results for the setup of MalGAN was a two hidden layer feed-forward network with sigmoid activation for each the generator and the discriminator. Each of the black boxes were fooled at the highest rate when running against this architecture of MalGAN

7 Conclusion

Overall, this project was a rewarding challenge. I was able to learn a lot about different machine learning algorithms and neural network architectures while applying into a real world scenario. This paper along with others that study similar topics open a huge door in the research world as to how malware detection systems that are comprised of machine learning algorithms can be strengthened against adversarial examples.