Evading Deep Neural Network and Random Forest Classifiers by Generating Adversarial Samples

Erick Eduardo Bernal Martinez[1], Bella Oh[2], Feng Li[3], and Xiao Luo[3(✉)]

[1] Department of CS, IUPUI, Indianapolis, IN, USA
ebernalm@iu.edu
[2] Department of CSE, Michigan State University, East Lansing, MI, USA
oheunjeo@msu.edu
[3] Department of CIT, IUPUI, Indianapolis, IN, USA
{fengli,luo25}@iupui.edu

Abstract. With recent advancements in computing technology, machine learning and neural networks are becoming more wide-spread in different applications or software, such as intrusion detection applications, antivirus software and so on. Therefore, data safety and privacy protection are increasingly reliant on these models. Deep Neural Networks (DNN) and Random Forests (RF) are two of the most widely-used, accurate classifiers which have been applied to malware detection. Although their effectiveness has been promising, the recent adversarial machine learning research raises the concerns on their robustness and resilience against being attacked or poisoned by adversarial samples. In this particular research, we evaluate the performance of two adversarial sample generation algorithms - Jacobian-based Saliency Map Attack (JSMA) and Fast Gradient Sign Method (FGSM) on poisoning the deep neural networks and random forests models for function call graph based malware detection. The returned results show that FGSM and JSMA gained high success rates by modifying the samples to pass through the trained DNN and RF models.

Keywords: Adversarial Machine Learning · Neural Network · Random Forest · Graphlet

1 Introduction

With recent advancements in machine hardware and computing technology, researchers have been focusing on expanding the usage of machine learning. Machine learning refers to inputting large amounts of data to an algorithm during a training phase. This, in turn, causes the algorithm (or model) to learn particular patterns of the training data, and then react on new data depending on those learned patterns. With this technique, machines gain the ability to self-improve and adapt to new data. This technology can be used in several different

N. Zincir-Heywood et al. (Eds.): FPS 2018, LNCS 11358, pp. 143–155, 2019.
https://doi.org/10.1007/978-3-030-18419-3_10

aspects, including related searches on Google or self-driven cars. Because of the automation gained from it, machine learning can increase the comfort and efficiency of our daily lives. Hence, this topic is currently under intensive studies. However, on the other side, hackers attempt to attack or poison the machine learning algorithms. Hacking into the machine learning algorithm leads to great security threats on data privacy and security; therefore, adversarial machine learning, a new research field intending to investigate how to detect adversarial activities of the machine learning algorithms, is needed.

In this paper, we investigate and compare two adversarial example generation algorithms that attempt to mislead the trained machine classifiers for malware detection. The first algorithm is known as Jacobian-based Saliency Map Attack (JSMA), proposed by Papernot et al. [8]. It exploits the forward derivative of a deep neural network (DNN) to find perturbations, then creates adversarial modifications to some components of normal malware data using the perturbations. The second algorithm is known as Fast Gradient Sign Method (FGSM) [12]. This method aims to generate malicious samples by using gradient computations. Both methods attempt to generate adversarial malware samples so that a trained machine learning model would classify the adversarial malware samples as benign. In this research, we explore and compare these two adversarial example generation algorithms on two well-known learning algorithms: deep neural network (DNN) and random forest (RF). Deep neural networks are a series of interconnected nodes. Each connection between nodes has a weight associated with it, and the nodes are structured in layers: an input layer, a series of hidden layers, and an output layer. The random forest, another machine learning algorithm, is composed of several decision trees. It classifies data by growing binary branches from splitting variables, and it is known to be useful for several classification and regression tasks. The malware dataset used in this research is a graphlet dataset which include 2394 function call graphs extracted from Android malware and benign applications. Each malware or benign instance in this dataset is described as a set of graphlets. This dataset is built by a previous research [3]. The returned results of this research show that FGSM can generate instances to evade both DNN and RF algorithms with 100% success rate when degree of perturbation is high. On the other hand, JSMA can gain high success rate with DNN but not with RF algorithm. We hypothesize that is because JSMA is based on selecting the features first before generating perturbation values; therefore, the selected features might be different from those selected by RF to build the tree structure. The FGSM can be further evaluated on generate instances to evade other machine learning algorithms in the near future.

The rest of paper is organized as following: related work is presented in Sect. 2, the details of the FGSM and JSMA algorithms for generating adversarial samples are given in Sect. 3, two classification algorithms - DNN and RF are described in Sects. 4 and 5 demonstrates the experimental settings and discusses the results, Sect. 6 explains how we validate the generated adversarial samples, the conclusion and future work is concluded in Sect. 7.

2 Related Work

Adversarial Machine Learning is a relatively new field of study. Attacks on Deep Neural Networks and Random Forests, in particular, are emerging. There are related techniques and concepts described in the literature.

Some literatures provide tactics of adversarial data manipulation given that the target classifier is minimally exposed to the adversary, such as having only the information of its final classification decision. Given these conditions, Dang et al. [13] performed an evasion attack on non-image datasets. They targeted two generic PDF malware classifiers - PDF_{RATE} and Hidost - then generated malicious samples using only a blackbox morpher and a sandbox in order to find the most optimal path of modifications needed for a file to missclassify from malware to benignware. This met the challenge of evading the two PDF classifiers by morphing malicious samples "in the dark". The method, although effective and capable of wider application, was limited by the amount of queries that can be made to a classifier while also avoiding detection. Another method of evasion is generating malicious samples and testing them on phishing website classifiers. Hu and Tan [13] proposed a generative adversarial network (GAN) based algorithm named MalGAN to generate adversarial malware examples. These examples were able to bypass machine learning based detection models, not knowing what type of model it is. This was made possible due to a substitute detector that MalGAN uses to fit the black-box malware detection system. Then, a generative network was trained to minimize the malicious probabilities predicted by the substitute detector. Although relatively difficult in application, MalGAN proves to be effective, decreasing adversarial malware detection rates to almost zero.

More recently, Elsayed, et al. [15] devised a technique to re-purpose a target model by means of an adversarial program. The adversarial program was added to a network's input in order to force it to perform a different task. This kind of attack was ideal for an adversary that has gained access to a neural network's parameters. Grosse et. al. [16] also constructed an adversarial attack on machine learning malware detectors. They take into account the existing adversarial-crafting algorithms, but expand on it to use discrete, often binary, input domains as opposed to continuous ones. In addition, they proved that their manipulated program would perform malware functions. This work provided a more discrete, yet promising method to generate adversarial samples. Jia et. al. [17], on the other hand, focused on the prevention of adversarial attribute inference attacks, specifically on machine learning algorithms. An attribute inference attack leverages a machine learning classifier to infer a target user's private attributes. To counteract this, Jia et. al. [17] created a method that evades users' original information from the attacker's classifiers by adding noise to the user information. This study showed a way of using evasion as security instead of attack. Another recent work looked into classifier evasion via altering malware binary payloads [18], which is the most similar to ours. It also aimed to evade neural network malware/benignware classifiers. However, it differs in that we also evaluate evasion success in RF classifiers and that our approach does not modify binary

payloads. Rather, we propose that an adversary can simply modify the function calls of their malware to attain the results we seek.

Although all of the mentioned works inspired our work, the strongest motivation behind this literature was to utilize the methods developed by Goodfellow et al. [12] and Papernot et al. [5]. Both of these works develop methods aimed to perturbing MNIST images in order to cause misclassification. Goodfellow et al. addresses a method known as *Fast Gradient Sign Method* (FGSM). Given a model with an associated cost function, the FGSM crafts an adversarial sample for an original sample by computing perturbations. Perturbations are made to every value of the whole sample vector. Increasing the perturbations increases the likelihood of the adversarial sample being mis-classified, but also decreases the discreteness of the adversarial change.

Papernot et al. [5] propose the *Jacobian-based Saliency Map Attack* (JSMA) respectively. Given a model, an adversarial sample is created by adding a perturbation to only a subset of the values in the given sample vector. The two attacks will be elaborated further in Sect. 5 of our paper. The JSMA attack is known to be strong with targeted misclassification and creates less perturbations than the FGSM, making it less detectable, but with greater cost. FGSM is more applicable in any misclassification attack and can quickly create many adversarial samples, but makes larger perturbations, creating danger of detection. After the proposal of these two attacks, Papernot et al. [9] then makes use of both methods to study the evading success of the attacks among different machine learning classifiers.

Our work differs from the above in that we attempt to use the FGSM and JSMA methods developed by Goodfellow et al. and Papernot et al. respectively in order to attack the trained DNN and RF classifiers using the malware Graphlet Frequency Distribution (GFD) vectors. We have also studied the perturbation degree required to attain satisfactory results as well as the evading success to Deep Neural Network and Random Forest classifiers.

3 Adversarial Sample Generation

Consider a malicious adversary that wants to deploy a malicious android application in a setting where their target uses a malware detection engine. The adversary has knowledge about the classifier used by their target, but does not have access to the model used by their target. Further, the adversary is aware that the target uses either a DNN or RF model to classify via graphlet frequency distributions of function call graphs. The adversary also has access to a modest set of malicious and benign android applications, and uses them to create a simple DNN or RF malware classifier.

In this study, we compare two algorithms to execute evasion attacks on classifiers that have completed their training phase. The goal of these algorithms is to modify malware instances so that the changes are unnoticeable, but they are enough to mislead the classifier into classifying the modified malware as benign. This section explains the two different attack methodologies used to maliciously modify the regular malware samples.

3.1 Fast Sign Gradient Method (FGSM)

The Fast Sign Gradient Method is developed by Goodfellow, et al. [12], which can be summarized follows. Given a classification model F with an associated cost function $J(\theta, x, y)$, the adversary generates an adversarial sample using Eq. 1:

$$\boldsymbol{x}^* = \boldsymbol{x} + \eta \tag{1}$$

where η is defined as Eq. 2:

$$\eta = \epsilon \; sign(\nabla_x J(\theta, x, y)) \tag{2}$$

where ϵ refers to a perturbation, which can be any arbitrary number between 0 and 1. This perturbation value influences the significance of the modification as well as the effectiveness of the algorithm. Higher ϵ values yield better results, however they also result in less subtle differences between $\boldsymbol{x}$ and $\boldsymbol{x}^*$. Nevertheless, the objective is to generate sample $\boldsymbol{x}^*$ to be mis-classified from class x to class y without modifying the model F. In our study, the objective is to generate $\boldsymbol{x}^*$ to be mis-classified from malware to benign without modifying the Deep Neuron Networks and Random Forest.

3.2 Jacobian-Based Saliency Map Attack (JSMA)

The Jacobian-based Saliency Map Attack (JSMA), proposed by Papernot et al. [8], exploits the forward derivative of a deep neural network (DNN) and finds perturbation. To decide which features to change, the algorithm orders each feature by its importance in creating the adversarial sample. Then, the input components are added to the perturbation value in the importance order. This is continued until the sample becomes mis-classified as the target. This is accomplished using the adversarial saliency map, defined as Eq. 3.

$$S(X,t)[i] = \begin{cases} 0, if \; \frac{\partial F_i(X)}{\partial X_i}) < 0 \; or \; \sum_{j \neq t} \frac{\partial F_i(X)}{\partial X_i}) > 0 \\ (\frac{\partial F_i(X)}{\partial X_i})|\sum_{j \neq t} \frac{\partial F_i(X)}{\partial X_i})|, otherwise \end{cases} \tag{3}$$

Given target t, exploiting this map allows $F_t(X)$, the target class output, to increase while probabilities $F_j(X)$ of other classes $j \neq t$ decrease, until target $t = arg \; max \; F_j(X)$. With this method, modifications can be made to malware data that will lead the model to mis-classify it as normal. The advantage of this algorithm is that it requires very small changes to be made to minimum number of features from the original malware data. Hence, it makes the attack more discrete. However, the computational cost is higher than the FGSM method.

4 Learning Algorithms

In this study, we evaluated the two adversarial sample generation algorithm on two well-known and widely used learning algorithms: Deep Neural Network and Random Forest. These two algorithms are described as following.

4.1 Deep Neural Network (DNN)

Deep Neural Network (DNN) is a machine learning technique consisting of interconnected nodes. An example of a DNN classifier is given in Fig. 1. There are several implementations of this technique. However, they all have three key features: one layer of input nodes, a set number of hidden layers, and a final layer of output nodes. Each connection from one node to another has a weight value; this value is used to determine the activation of the node it is connected to. The number of nodes and connections in each layer are ideally set to represent some linear or non-linear problem; the network as a whole is called the model. In this study, we implement a fully connected neural net as a classifier for detecting malware in the dataset.

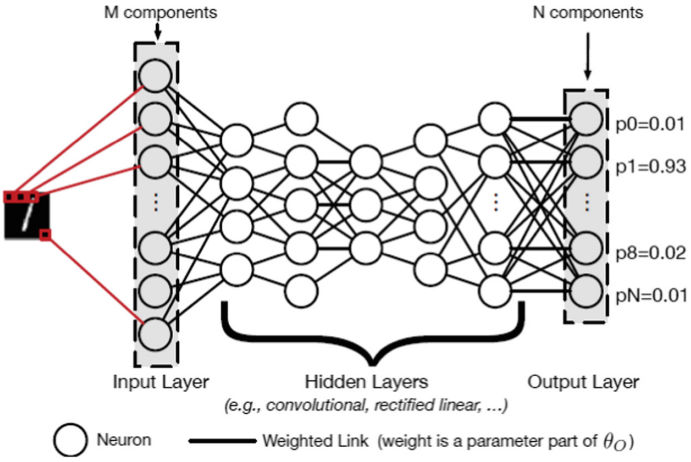

Fig. 1. Example of a DNN classifier. M components are the number of neurons at the input layer. Normally, they equal to the number of features in the dataset. N components at the output layers correspond to the number of classes to predict. Given an input, there is a associated probability of each class at the output layer. The probability to a class is between 0 and 1.

The Deep Neural Network is similar to other machine learning algorithms. In order to build the model, a training phase and a testing phase are involved. During the training phase, the network accepts inputs and feed forward mathematical operations that ultimately set off an activation function on each node of the network. The activation of the node relies on the weights associated with itself. Normally, higher weight results in greater activation. The activation of the final layer is then compared to the inputs, and a cost or a loss is computed. The weights of the final layer and the preceding layers are adjusted based on the cost function in order to minimize the cost of the next prediction. Once the training phase is done, the trained model can be loaded into a system for testing. The testing phase simply refers to inputting test data that was not used in the train

phase and then predicting their class labels - malware or benign in the study, and calculating the accuracy.

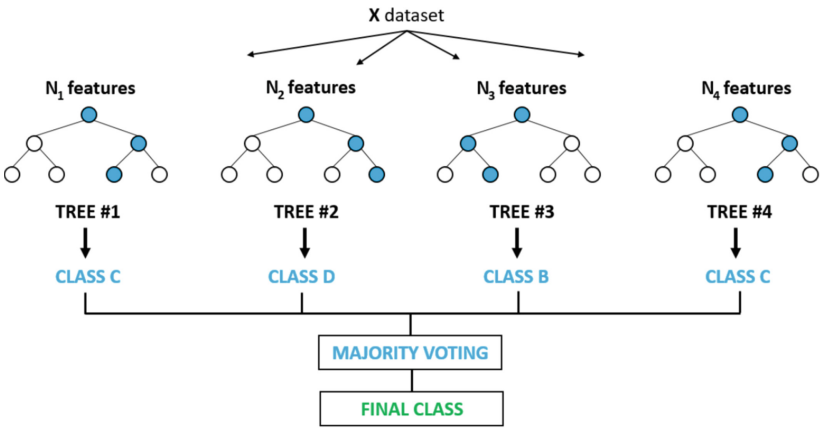

Fig. 2. RF classifier and its basic structure.

4.2 Random Forest (RF)

A random forest (RF) is a classifier composed of several decision trees [4]. Figure 2 demonstrates a visual explanation of the random forest skeleton [14]. If a dataset D is the input of the classifier, the algorithm creates several sets of randomly selected samples of D. Each decision tree randomly takes one set of the samples as its input. Then, each tree grows binary branches from splitting variables. For this study, the trees use the entropy algorithm to determine splitting variables. Entropy measures the uncertainty of the class in a subset of examples, and the goal is to maximize the purity of the groups for every split. Therefore, the algorithm will always choose a splitting variable that will decrease the entropy. Each tree stops growing either after reaching maximum depth or purity. After all decision trees classification finish, the most frequent output (majority vote) of all the trees becomes the output of the random forest. Comparing to decision tree, the random forest may be visually difficult to interpret. However, it's supposed to avoid the problem of over-fitting due to the "majority vote" output. Also, it increases independence among trees, making it robust when dealing with a large number of features in data.

5 Experiments and Results

5.1 Data Set

The dataset used in this study is a graphlet dataset for function call graphs of Android applications [3]. It is comprised of 2394 samples. The training set

consists of 597 malware and 600 benign samples, while the testing set consists of 600 malware and 597 benign samples. These malware and benign samples present function call graphs of programs that are extracted from the Android applications [10]. A function call graph contains nodes and edges, each node represents a function while each edge represents a function call. Each function call may be one way or bi-directed. A graphlet is a sub-graph which consists of n nodes from a function call graph. Figure 3 shows the number of combinations obtained from a graphlet of $n = 3$. A function call graph can contain different combinations of these graphlets. Gao et al. [10] extracts graphlet frequency distribution (GFD) vectors from $n = 3$ up to $n = 5$. A GFD vector presents a function call graph. Each entry of the vector presents the frequency of a type of a graphlet i ($i \in \{1, \ldots, N\}$). In this dataset, there are $N = 125$ different types of graphlets. They are attributes of the input data to train and test the DNN and RF classifiers.

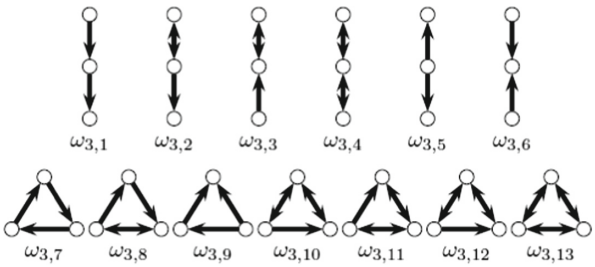

Fig. 3. The 13 graphlet types when $n = 3$

5.2 Classification Performance on Normal Samples

Before the learning algorithms are tested by the generated adversarial samples, they are evaluated and compared against the traditional evaluation metric for classification. Since only malware and benign samples are included in our experimental dataset, this turns to a binary classification task.

Deep Neural Network. The deep neural network implemented in this research consists of 125 input nodes which corresponds to the types of graphlet, 3 hidden layers consisting of 100, 50 and 10 nodes respectively, and 2 output nodes which corresponds to the output classes: malware and benign. The training set described in Sect. 5.1 is used to train the model, then the model is tested by the test set. After the network is trained for 15 epochs utilizing an Adam optimizer and a learning rate of 0.0001, it gains an accuracy 85.38% training set and 83.62% on test set.

Random Forest. The "Random Forest" package in WEKA machine learning tool [11] is used to implement the algorithm. The network has 125 input nodes, one node for each graphlet type. Same training and test data setting is used for training the RF model. The RF algorithm achieves 99.9% on the training set and 87.80% accuracy on test set. Although the accuracy rates of RF algorithm are higher than those gained by DNN algorithm, the accuracy differences between training and testing is significantly higher than that of the DNN algorithm. This implies that the RF algorithm might overfit to the training data.

Table 1 provides an overview of the accuracy of both algorithms.

Table 1. Classification accuracy on the original data

	DNN	RF
Training	85.38%	99.9%
Test	83.62%	87.80%

5.3 Performances of Adversarial Sample Generation Algorithms

In this section, we analyze and output the overall effectiveness and efficiency of the adversarial sample generational methods - FGSM and JSMA.

The two methods are evaluated from two aspects: (1) the average difference between original malware and generated adversarial malware with the increasing of the perturbation (a parameter that controls the subtlety of modifications made) (2) the relationship between perturbation and the success rate of evading the detection of the trained machine learning models. In order to evaluate two methods from the first aspect, 200 malware samples from the testing dataset were modified through using FGSM and JSMA respectively. Figure 4 shows the

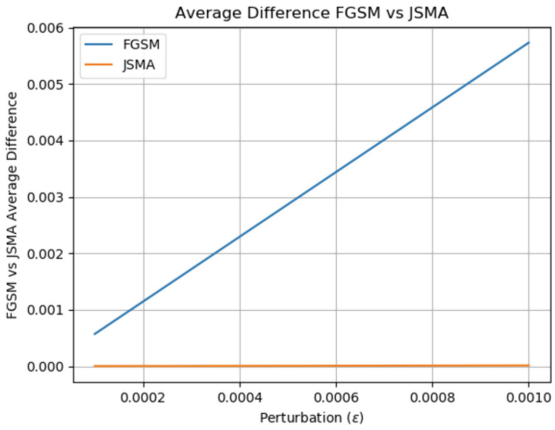

Fig. 4. Average difference among adversarial vs. original GFD vectors

averaged difference of the 200 samples with the increase of perturbation. The averaged difference is calculated as the sum of element-wise difference between the original malware GFD vector and the generated adversarial malware GFD vector. It is found that as the perturbation increases, the difference between the original malware and generated adversarial malware by FGSM increases, whereas the difference between the original malware and the generated adversarial malware by JSMA is not noticeable and stays almost the same. Through looking into the detailed adversarial malware, it is found that JSMA only makes very negligible changes to most of the original malware graph. It increases a few values from 0 to whatever the perturbation was. For example, if the perturbation value is .001, then it typically changes about 2 elements of the GFD from 0 to the perturbation and leaves the rest nearly identical.

After analyzing the degree of perturbations and their effects, we generate 5 groups, a total of 1000 adversarial generated malware using perturbation value of $\epsilon = 1e-7, 5e-7, 1e-5, 5e-5$, and $1e-3$ respectively. Then, these groups of 200 adversarial malware are inputted into the trained DNN and RF classifiers. The success rate is used to evaluate how success the generated adversarial malware can evade the trained DNN and RF. Success rate is calculated as Eq. 4. The total number of adversarial malware is 200 in this case.

$$Success\ Rate = \frac{Number\ of\ adversarial\ malware\ classified\ as\ benign}{Total\ number\ of\ adversarial\ malware} * 100. \tag{4}$$

The success rates for all 5 groups are shown in Table 2 and Fig. 5. From the results, we can tell the FGSM attack creates successful adversarial examples for both classifiers with good success rates when the perturbation is about 1e-5. In order to gain a high success rate, JSMA needs a higher perturbation value than the FGSM. We also notice that the FGSM-generated adversarial malware can evade the detection of both DNN and RF. Meanwhile, the JSMA-generated adversarial malware cannot evade the detection of RF. Only 13–14% of the adversarial malware generated by JSMA are classified as benign. We hypothesize that this is due to the nature of the algorithms of the RF and JSMA. The JSMA only modifies around 10 to 20 most important attributes of the input. Given that its attribute importance ranks are similar to that of the RF, slightly changing important attributes might only change some of the top few splitting variable attributes of the RF. If the change is not significant, this may not change the RF's decision. To improve the success rate of the JSMA could be increasing the number of features to modify. However, it might be computationally costly.

Overall, both adversarial malware generation methods successfully generate malware that evade the detection of DNN. Based on Table 2, given that the success rate of JSMA on RF is relatively low, we could draw the conclusion that RF may be a more robust model for classifying malware presented as GFD vectors. However, this might also be because the JSMA method creates adversarial malware using neural network features (number of inputs, prediction outputs, etc.), and are therefore more specifically geared to target neural networks like DNN.

Table 2. Different Model Performance Results

Attack method	Performance on attack (%)		
	Perturbation	RF	DNN
FGSM	1.e-7	13.0	16.5
JSMA	1.e-7	13.0	12.5
FGSM	5.e-7	14.5	24.0
JSMA	5.e-7	13.0	12.5
FGSM	1.e-5	81.0	92.5
JSMA	1.e-5	13.0	13.5
FGSM	5.e-5	100.0	100.0
JSMA	5.e-5	14.0	89.5
FGSM	1.e-3	100.0	100.0
JSMA	1.e-3	14.0	100.0

6 Conceptual Validation of the Generated Adversarial Malware

In this section, we demonstrate how a function call graph can be modified conceptually to reflect generated adversarial malware - graphlet frequency distribution (GFD) vector. A function call graph can be sub-divided into graphlets which consists of n nodes. Figure 3 demonstrates 13 directed graphlets of 3 nodes that could be found in a FCG. The GFD vector is a vector of length m which is the total number of different graphlets in the whole dataset. Each unit of the vector

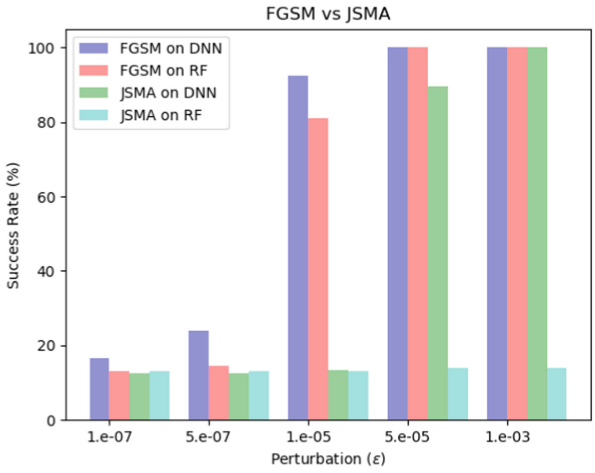

Fig. 5. Comparison between FGSM and JSMA performance on DNN and RF classifiers

is the relative frequency of the corresponding graphlet in the function call graph [10]. The relative frequency of each type of graphlet is calculated as Eq. 5:

$$\frac{f_{n,i}}{\sum_{i=1}^{n} f_{n,i}} \tag{5}$$

where $f_{n,i}$ is type i graphlet of n nodes,

Hence, one relative frequency of a graphlet increases, it affects the relative frequencies of every other graphlet. So, a desired adversarial GFD vector is minimized change to the units of the original GFD vector. As shown in Fig. 5, both FGSM and JSMA make small changes to the GFD vectors. In particular, JSMA maintains the changes needed in order to attain desired results to only modifying but a few units of the original GFD vector. In our test, JSMA simply increases the frequency of a unit from zero to the value of the assigned perturbation (ϵ) while making negligible changes to other units. We have investigated how many units increased in this manner by using ϵ values of $1e-5, 1e-4, 1e-3, 1e-2, 1e-1$. It is found that 10, 4, 2, 2 and 2 units changed with these ϵ values respectively. This indicates that the less number of units affected the more ϵ grows. So, in order to change the original malware to adversarial one, a very small amount of graphlets to original function call graph is needed. More specifically, $\epsilon \sum_{i=1}^{n} f_{n,i}$ need to be added to the original function call graph of the malware. The changes are small enough that the generated adversarial GFD vector remains nearly identical to the original.

7 Conclusion and Future Work

In this research, we explore two algorithms: FGSM and JSMA for generating of adversarial graphlet frequency distribution (GFD) vectors. The Deep Neural Network and Random Forest classifiers are used to test adversarial graphlet frequency distribution (GFD) vectors to see whether they can evade the detection. It is found that the FGSM gains 100% success rate on both DNN and RF classifiers by using any perturbation $\epsilon \geq 1e-5$, while the JSMA samples yields high success rates (close to 100%) when using $\epsilon \geq 1e-4$ on classifier DNN. The JSMA method on the RF classifier merely succeeds in misclassifying 14% of samples with perturbations as high as $\epsilon = 1e-3$. Nevertheless, we conclude that both methods are a viable way to generate adversarial graphlet frequency distribution vectors. In the future, we plan to evaluate FGSM and JSMA on other machine learning algorithms and other types of data. Integration of FGSM and JSMA algorithms can also be investigated in the future.

Acknowledgment. This research was made possible with the support of the Indiana University-Purdue University Indianapolis Department of Computer Information and Information Science, with funding from National Science Foundation and United States Department of Defense. The author(s) would like to thank to Dr. Mohammad Al Hasan, Tianchong Gao and Sheila Walter for their support.

References

1. Hu, W., Tan, Y.: Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN (2017)
2. Subasi, A., Molah, E., Almkallawi, F., Chaudhery, T.J.: Intelligent phishing website detection using random forest classifier. In: 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, pp. 1–5 (2017)
3. Peng, W., Gao, T., Sisodia, D., Saha, T.K., Li, F., Al Hasan, M.: ACTS: extracting android app topological signature through graphlet sampling. In: 2016 IEEE Conference on Communications and Network Security (CNS), pp. 37–45 (2016)
4. Bosch, A., Zisserman, A., Munoz, X.: Image classification using random forests and ferns. In: 2007 IEEE 11th International Conference on Computer Vision, Rio de Janeiro, pp. 1–8 (2007)
5. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., and Swami, A.: The limitations of deep learning in adversarial settings. In: Proceedings of the 1st IEEE European Symposium on Security and Privacy, pp. 372–387 (2016)
6. Feinman, R., Curtin, R.R., Shintre, S., Gardner, A.B.: Detecting Adversarial Samples from Artifacts. arXiv preprint arXiv:1703.00410 (2017)
7. Papernot, N., Carlini, N., Goodfellow, I., Feinman, R.: Cleverhans v2. 0.0: an adversarial machine learning library. arXiv preprint arXiv:1610.00768 (2016)
8. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. IEEE European Symposium on Security and Privacy (EuroS, P), Saarbrucken, pp. 372–387 (2016)
9. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS 2017), pp. 506–519. ACM, New York, (2017), https://doi.org/10.1145/3052973.3053009
10. Gao, T. et al.: Android Malware Detection via Graphlet Sampling, pp. 1–14 (2018). Unpublished
11. The WEKA Workbench: Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", 4th edn. Morgan Kaufmann (2016)
12. Goodfellow, I.J., et al.: Explaining and harnessing adversarial examples. In: Proceedings of the International Conference on Learning Representations (2015)
13. Dang, H., Huang, Y., Chang, E.: Evading classifiers by morphing in the dark. In: ACM CCS, pp. 119–133. ACM (2017)
14. Holczer, B.: Random Forest Classifier - Machine Learning. Global Software Support, 7 March 2018. www.globalsoftwaresupport.com/random-forest-classifier-bagging-machine-learning/
15. Elsayed, G., Goodfellow, I., Sohl-Dickstein, J.: Adversarial Reprogramming of Neural Networks (2018). https://arxiv.org/pdf/1806.11146.pdf. Accessed 22 Oct 2018
16. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 62–79. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_4
17. Jia, J., Gong, N.Z.: AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning. https://arxiv.org/pdf/1805.04810.pdf. Accessed 22 Oct 2018
18. Kreuk, F., et al.: Deceiving End-to-End Deep Learning Malware Detectors using Adversarial Examples (2018). https://arxiv.org/pdf/1802.04528.pdf. Accessed 22 Oct 2018