2019 IEEE International Conference on Cognitive Computing (ICCC)

Securing Malware Cognitive Systems against Adversarial Attacks

Yuede Ji, Benjamin Bowman, H. Howie Huang
George Washington University
{*yuedeji, bbowman410, howie*}*@gwu.edu*

***Abstract*—The cognitive systems along with the machine learning techniques have provided significant improvements for many applications. However, recent adversarial attacks, such as data poisoning, evasion attacks, and exploratory attacks, have shown to be able to either cause the machine learning methods to misbehave, or leak sensitive model parameters. In this work, we have devised a prototype of a malware cognitive system, called DEEPARMOUR, which performs robust malware classification against adversarial attacks. At the heart of our method is a voting system with three different machine learning malware classifiers: random forest, multi-layer perceptron, and structure2vec. In addition, DEEPARMOUR applies several adversarial countermeasures, such as feature reconstruction and adversarial retraining to strengthen the robustness. We tested DEEPARMOUR on a malware execution trace dataset, which has $12,536$ malware in five categories. We are able to achieve 0.989 accuracy with 10-fold cross validation. Further, to demonstrate the ability of combating adversarial attacks, we have performed a white-box evasion attack on the dataset and showed how our system is resilient to such attacks. Particularly, DEEPARMOUR is able to achieve 0.675 accuracy for the generated adversarial attacks which are unknown to the model. After retraining with only 10% adversarial samples, DEEPARMOUR is able to achieve 0.839 accuracy.**

***Keywords*-cognitive; machine learning; malware; adversary;**

I. INTRODUCTION

A cognitive system is self-learning by leveraging a combination of intelligent techniques, such as machine learning (ML), data mining [1]. The cognitive systems along with the machine learning techniques have achieved great progress in recent years. They have provided breakthrough performance across many domains such as image processing [2], self-driving vehicles [3], and cybersecurity [4].

Recent studies find that many of the cognitive systems are vulnerable to adversarial attacks [5]. In essence, adversarial attacks try to cause the machine learning methods to misbehave or leak sensitive model information, and can take place throughout different stages of learning. In the training stage, data poisoning attack injects incorrectly or maliciously labeled samples into the training dataset to make the machine learning methods learn incorrectly [6]. In the testing stage, evasion attacks tamper with test data to cause prediction errors. In addition, exploratory attacks will repeatedly test the learned model with edge-cases to reveal the decision boundary [7]. In this paper, we are specifically focusing on evasion attacks as these are the most common attacks on machine learning models. Different adversarial defense techniques have been proposed [8] [9]. However, most of them are targeting the adversarial attacks in the computer vision problem [10]. Also, most defending techniques are only effective for a few attacks which are usually known to the designer in advance [10].

Malware cognitive systems, which apply cognitive intelligence to malware detection, have gained great popularity rencently. A security company, Sparkcognition, designs a cognitive agent to better detect various types of unknown malware [11]. The network company, Cisco, leverages cognitive intelligence to better defeat polymorphic malware [12]. Further, the enterprise cognitive systems (ECS) have been used by several companies, i.e., IBM [1], Cybereason [13], towards better malware detection. However, recent studies have shown that existing malware cognitive systems are vulnerable to data poinsoning adversarial attacks [14]. Motivated, In this work, we explore the adversarial defense for the malware cognitive system with the special focus on the unknown adversary. Specifically, the dataset[1] is the bag-of-n-grams [15] extracted from the malware API call sequence. The adversary will be generated from the bag-of-n-grams.

To address this problem, we design DEEPARMOUR, a robust malware classification system against unknown adversarial evasion attacks. DEEPARMOUR consists of three adversarial defense techniques, namely feature reconstruction, weighted voting, and adversarial retraining. Given the bag-of-n-grams of a malware, DEEPARMOUR will reconstruct the features in two ways: term frequency-inverse document frequency (TFIDF) weighting, as well as a graph representation. Next, DEEPARMOUR votes on the label via three classifiers: random forest, multi-layer perceptron (MLP), and structure2vec. To strengthen the robustness, we automatically generate some targeted adversarial evasion attacks against DEEPARMOUR and retrain them.

In summary, we make the following contributions.

- We have designed two feature reconstruction methods, term frequency-inverse document frequency (TFIDF) weighting, and the attributed graph representation. TFIDF weights the importance of an API to a malware. The attributed graph representation provides a new way

[1]The dataset is from AAAI-19 Workshop on Artificial Intelligence for Cyber Security (AICS) Challenge problem (http://www-personal.umich.edu/~arunesh/AICS2019/index.html).

978-1-7281-2711-8/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCC.2019.00014

to clean the perturbations of adversarial samples. When testing with unknown adversarial attacks, the reconstructed features greatly increase the performance, i.e., the accuracy improves from 0.118 to 0.675, the macro F1 improves from 0.075 to 0.461, and the weighted F1 improves from 0.155 to 0.665.

- The weighted voting model is built on top of three classifiers: random forest, multi-layer perceptron, and structure2vec, each of which uses different learning strategy as well as different features. As a result, this model can mitigate the risk of one classifier being fooled by an adversary by taking the majority vote across all classifiers. The tests on the unknown adversarial attacks show the effectiveness of voting. Particularly, compared with the best single classifier, structure2vec, the voting improves the accuracy from 0.66 to 0.675, the macro F1 from 0.223 to 0.806, and the weighted F1 from 0.649 to 0.806.
- We have implemented DEEPARMOUR and tested on various dataset. Particularly, DEEPARMOUR is able to achieve 0.989 accuracy, which is the average of 10-fold cross-validation on the normal malware dataset. Testing with our generated adversarial samples, which are unknown to our model, DEEPARMOUR is able to achieve 0.675 accuracy. By retraining only 10% adversarial samples, DEEPARMOUR is able to achieve 0.839 accuracy for the remaining adversarial samples. When retraining with 50% adversarial samples, DEEPARMOUR can reach 0.904 accuracy.

II. BACKGROUND

In this section, we present the background of adversarial attack and malware classification.

A. Adversarial Attack

Adversarial attacks against machine learning can be categorized into the following three types:

Evasion attacks are the most common types of adversarial attack against ML algorithms. An adversary performs evasion attacks by manipulating inputs to a trained model with the goal of producing incorrect output. There are many approaches to generating these samples. Typical approaches perform so-called white-box attacks as the parameter weights of the machine learning model are known. This allows an attacker to identify the network gradients and thus perform targeted manipulation of the inputs to produce an erroneous output [5] [16].

Data poisoning attack assumes access to the training phase of the model. An adversary performs such an attack by contaminating the training data in a certain way to produce incorrect output at inference time. Typically the attack scenario involves finding the minimal amount of contamination required to maximize the intended degradation of the targeted model [17] [6].

Exploratory attacks differ from the previous two as they are not directly attempting to fool the network in a specific way, but rather these attacks aim to extract knowledge from a trained network not explicitly provided by the model owners. For example, successful adversary attacks have extracted the models themselves [18], as well as have inferred information about the training and testing data used to train the model [7]

In the context of malware classification, we focus on the evasion attacks as it is assumed that our training data is free of adversarial input, and we are not providing our model directly to the adversary. Instead, we are provided a set of training samples with the indication that some of the samples will be adversarial in nature.

B. Malware Classification

Malware is software with malicious intent such as crashing the infected device, stealing user information, and launching phishing attacks [19]. Malware classification is one of the major countermeasures for malware detection. It includes three different classification tasks. The most basic approach is a simple binary-class classification between malware and benign software. After a malware has been identified, it may be important to determine what strain of malware this sample belongs to. To accomplish this, one can do multi-class classification of the malware types, such as virus, worm, Trojan, and botnet [20]. For each malware type, one can further classify them into different malware families [21]. In this project, we are focusing on the malware classification in terms of malware types.

III. PROBLEM DEFINITION

In this section, we define the task and threat model.

A. Task Definition

This project aims at defending adversarial attacks in the context of using machine learning for malware classification. There are five malware classes: Virus, Worm, Trojan, Packed Malware, and AdWare. There is no benign software involved. Each malware is dynamically analyzed on an Windows virtual machine and the sequence of Windows API calls is extracted. Further, each sequence is obfuscated by extracting bag-of-n-grams ($n = 1, 2, 3$) and each API is anonymized to an integer number. In other words, the real API and execution sequences are not provided. We try to design a robust malware classification system that can defend adverarial evasion attacks.

B. Threat Model

We define the adversarial threat model as follows: (1) The adversarial attacks can only happen at the testing stage. That means, the attacks targeting the training dataset, such as data poisoning attacks, will not be considered in this project. (2) The adversaries may have knowledge of our training dataset, but are not allowed to modify it. Our training dataset is

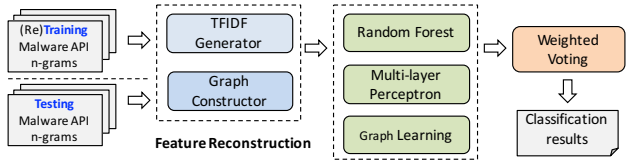

Figure 1. The architecture of DEEPARMOUR

exposed to the adversaries and they may analyze it to better attack our model. (3) The adversaries have no knowledge of our trained model (architecture, parameters), in other words, our model is a black-box to them. Although the adversarial examples we generated are based on our trained model, the final testing adversarial attacks have no knowledge of our trained model. (4) The adversaries only aim at degrading the performance in terms of accuracy metrics and are not attacking any confidentiality or privacy issues.

IV. OVERVIEW

In this section, we present an overview of DEEPARMOUR in terms of training and testing, as illustrated in Figure 1.

A. Training Stage

In this stage, DEEPARMOUR automatically learns a weighted voting malware classifier. Firstly, given the training dataset in the format of bag-of-n-grams extracted on the anonymized API execution sequence, DEEPARMOUR reconstructs the features in two ways. One converts the absolute frequency value into term frequency-inverse document frequency (TFIDF) value, which reflects the importance of each API call. The other constructs graphs from the unigram and bigram. The purpose of feature reconstruction is to clean the perturbations brought in by the adversarial samples [9].

Secondly, DEEPARMOUR builds a voting model with three classifiers, i.e., random forest, multi-layer perceptron, and a graph learning method named structure2vec [22]. The three classifiers are using different learning strategies and different features. Such a design is motivated by the observation that most effective adversarial attacks are targeting one or one type of machine learning method [23]. The voting mechanism can mitigate the risk of one classifier being fooled by the adversarial by weighted voting.

Thirdly, we leverage the adversarial retraining to strengthen the robustness of DEEPARMOUR which is one of the most effective adversarial countermeasures [8]. Particularly, we automatically generate many adversarial samples in the same format of training dataset. We retrain the adversarial samples by following the same training stage. Moreover, the voting weights will also be optimized by the adversarial samples.

B. Testing Stage

In the testing stage, DEEPARMOUR takes the unlabeled malware API n-grams as input and outputs its classification label. Following the same flow in the training stage, DEEPARMOUR firstly reconstructs the features with TFIDF generator and graph constructor. Then, DEEPARMOUR loads the three trained classifiers and gets their prediction probabilities. Finally, DEEPARMOUR predicts the malware class based on the weighted voting.

V. DESIGN

In this section, we present the key designs of DEEPARMOUR in defending the adversarial attacks, including feature reconstruction, weighted voting, and adversarial retraining.

A. Feature Reconstruction

Feature reconstruction is designed to clean the perturbations brought by outliers and adversarial samples. Given the bag-of-n-grams, DEEPARMOUR uses two different feature reconstruction methods, TFIDF and graph representation.

Term frequency-inverse document frequency (TFIDF) is a weighting factor for text mining, which is intended to show the importance of a word to a document in a large corpus [24]. The malware classification problem shares the similar context to text mining because we can regard a malware as a document and an API call as a term. Given the absolute frequency of each word, the TFIDF weighting will decrease the value of the words that frequently appear in many document types, such as "the", "a", and "an". At the same time, it will increase the value of those words that only appear in a limited number of document types or documents. The insight behind such a design is that if a word frequently appears in different types of documents, it will be less effective in classifying the documents. The analogy to malware classification is that those API calls which occur in all malware samples are likely to not be indicative of malware at all, and may in fact be common API calls to all windows executables. Thus it makes sense to consider such APIs with less weights than those that happen less frequently. Therefore, we use the TFIDF value instead of the absolute frequency to classify the malware.

TFIDF weight is computed by the following three equations. Firstly, we get the term frequency (TF) by calculating the relative frequency of a term t in a document d using Equation 1, where $f_{t,d}$ denotes the absolute frequency of term t in the document d, $\sum_{t' \in d} f_{t',d}$ denotes the accumulated frequency of all the terms in the document d.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

Next, we calculate the inverse document frequency (IDF) by following Equation 2, where $|D|$ denotes the number of total documents in the corpus, $td(t,D)$ denotes the number of documents in which a term t appears.

$$\begin{aligned} idf(t,D) &= log\frac{|D|}{1+td(t,D)} \\ td(t,D) &= |d \in D, \exists t \in d| \end{aligned} \tag{2}$$

Finally, the TFIDF value of a term t in the document d is calculated by multiplying its TF and IDF value as shown in Equation 3.

$$tfidf(t,d) = tf(t,d) * idf(t,D) \tag{3}$$

A **graph representation** keeps the structure information, which is an effective representation used by many malware detection methods [25]. By representing each malware as a graph, malware classification is transformed to a graph classification problem. Given the bag-of-n-grams, one can build a malware API graph by taking each API as a node and the bigram as an edge. However, such a design still faces two challenges.

Challenge #1: how to embed node ID. The node ID has a meaning in malware classification scenario and should be a part of the initial node embedding. Realizing the node ID is a categorical feature, a one-hot embedding should be a good fit. However, for the given problem in this work, there are $1,111$ different node IDs which would lead to a dimensionality explosion [26]. To solve this issue, we utilize the feature hashing technique [27], and are able to use 10-dimension embedding vector to represent each node ID.

Challenge #2: how to keep the information beyond the structure. While graphs keeps rich structural information [28], [29], a graph learning method requires more information to learn an accurate embedding for classification purpose. To this end, we build an initial embedding for each node as *[node_id, node_freq, out_edge_freq, in_edge_freq]*. For each node embedding, node_id is a 10-dimension embedding for the node ID, node_freq is the frequency value of the unigram, out_edge_freq is the average out_edge frequency value getting from the bigram, in_edge_freq is the average in_edge frequency value getting from the bigram. We do not use trigram because it has already been represented by connecting two bigrams in the graph.

B. Weighted Voting

Observing that most adversarial attacks are targeting one or one type of machine learning method [23], we design a weighted voting system by combining three different classifiers: random forest (RF), multi-layer perceptron (MLP), and structure2vec. RF and MLP are using the weighted TFIDF value, while structure2vec is using the graph representation.

DEEPARMOUR takes the prediction probabilities of the three classifiers as inputs, instead of their final prediction labels. This design is motivated by the fact that targeted adversarial attacks try to push a sample to cross the decision boundary between the original and targeted class. Although the adversarial sample will be detected as the wrong target class, the original class usually has the highest probability among the remaining classes [30].

In the training stage, DEEPARMOUR takes three probability vectors as the input and learns three parameters α, β, γ to minimize the loss value between the voted and actual label. The learning process is based on Equation 4 as follows:

$$\arg\min_{\alpha,\beta,\gamma} ||\alpha * Y_1 + \beta * Y_2 + \gamma * Y_3 - Y|| \tag{4}$$

where $\alpha+\beta+\gamma = 1$, Y_i denotes the 5-dimension probability vector of the i-th classifier, and Y is the actual label. The parameters are learned from both the normal training and the adversarial retraining samples. In the testing stage, DEEPARMOUR uses the three learned parameters and predicts the label as the one with the highest probability.

Random Forests is an ensemble method using many decision trees as weak learners. Random forest is a popular choice for many learning tasks as they typically perform very well and have a small number of hyper parameters to choose from. Our RF learner was configured with 100 trees and utilized the standard $\sqrt{d}$ random features for each split, where d is the total number of features available.

A fully-connected neural network (FCNN), or multi-layer perceptron (MLP) is the most traditional neural network architecture. It consists of an input layer, an output layer, and some number of hidden layers. We utilized a MLP with two hidden layers, each with 160 neurons, relu activation function, 0.001 learning rate.

Our last learner is a graph based deap learning method structure2vec [22]. This learner is able to build vector representations for graph nodes, subgraphs, or the entire graph structure, by learning embeddings for every node, as well an aggregation function to combine embeddings and capture relevant relationships. We set the intermediate and final embedding size as 128, iteration depth as 5, learning rate as 0.0001, epoch as 100.

C. Adversarial Retraining

Adversarial retraining is one of the most effective ways to defend known adversarial attacks. We firstly generate adversarial samples for this problem. Later, we retrain the three classifiers and the voting parameters.

Based on our threat model, we are only considering the adversary who is capable of evasion attacks. In the general case, it would likely entail modifying a malware sample to utilize different API calls to accomplish the same task. For example, the Virus could potentially be misclassified as AdWare. Because we do not have access to the original malicious applications, or the API mappings, we can only generate our adversarial samples based on the provided training data.

In this work, we implemented an effective adversarial sample generation method based on the Fast Gradient Sign Method (FGSM) [5], which relies on knowledge of the learned parameters within a neural network in order to build the adversarial samples. The task of generating the samples is very similar to how the nerual network is trained, except rather than updating the weights to minimize the loss, we

Table I
DATASET SPECIFICATION.

	Virus	Worm	Trojan	Packed Malware	AdWare	Total
Normal malware	11,844	11,253	771	692	512	12,536
Generated adversarial	1,303	308	120	111	87	1,929

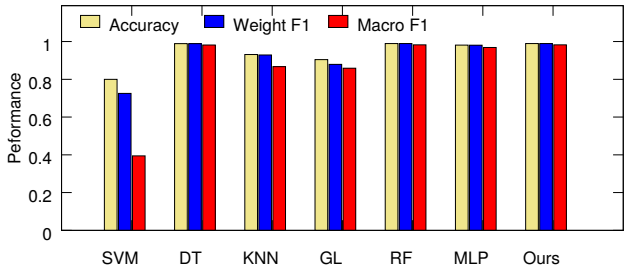

Figure 2. The performance on the non-adversarial dataset.

modify the input to minimize the loss. The loss in the targeted FGSM variant [31] is the difference in the predicted output and the target output. The adverserial samples are generated in an iterative approach described by the following equation:

$$X_*^0 = X; \\ X_*^{n+1} = Clip_X(X_*^n - \epsilon * sign(\nabla_x J(X_*^n, y_t)) \quad (5)$$

Here, X is our targeted training sample and X_* is our adversarial sample, y_t is our target label, ϵ is the perturbation parameter per iteration. The function J represents the cost function of the model, and ∇_x computes the gradient. The function $sign$ returns the sign of the resulting gradient vector. The $Clip$ function is an element-wise clipping function to constrain the features to proper values. In the context of malware execution traces, we define our clipping function to pin features to strictly integers greater than or equal to zero. We also set the perturbation parameter ϵ to 1.

VI. EXPERIMENT

The experiments are performed on a server with Intel Xeon E5-2620 (2.00 GHz) CPU, which has 12 cores with 15 MB of last-level cache and 128 GB of main memory. The server is equiped with one Nvidia Tesla K40c GPU which has 12GB memory. The server runs Ubuntu Linux (16.04) operating system. We use the machine learing library scikit-learn (version 0.19.1) and neural network framework TensorFlow (version 1.11.0).

The performance metrics we used are accuracy, weighted F1, and macro F1. The accuracy is calculated by dividing the number of correctly classified samples over the total number of samples. The basic F1 score is the harmonic average of the precision and recall. For multiclass evaluation,

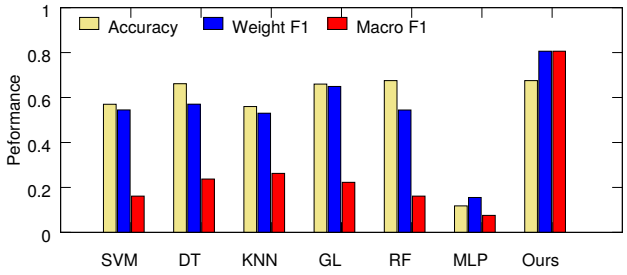

Figure 3. The performance of all the methods on the adversarial dataset.

the macro F1 calculates the metrics for each label and gets the unweighted mean, while weighted F1 gets the weighted mean based on the label imbalance.

A. Tests on the Normal Malware Dataset

We perform the baseline tests on the normal malware dataset, which is free of adversary attacks. The specification of this dataset is shown in Table I. We randomly split the whole dataset into ten groups and perform 10-fold cross validation. Besides the three classifiers in our voting system, random forest (RF), multi-layer perceptron (MLP), and structure2vec (GL), we also compare with support vector machine (SVM) with radial basis function (RBF) kernel, decision tree (DT), and k-nearest neighbor (k-NN, $k = 5$). All the classifiers are using the original features to train and test. In this experiment, DEEPARMOUR only applies the voting mechanism on the three clssifiers, not using the TFIDF weight or retraining.

The performance of 10-fold cross validation is shown in Figure 2, where we use the average of the 10 runs to represent. DEEPARMOUR and RF achieve the best performance in terms of accuracy, weighted F1, and macro F1. For the normal cases, DEEPARMOUR weights RF highest, MLP next, and GL lowest. For the accuracy, RF and DEEPARMOUR get the highest score 0.9894, DT gets 0.9888, MLP gets 0.981, GL gets 0.9045, and SVM gets 0.7998. Similar ranks are also observed in weighted F1 and Macro F1. Our method achieves high accuracy because it is voted on three classifiers, i.e., RF, MLP, and GL, and all of them achieve high accuracy.

B. Tests against Adversarial Attacks

To test the robusteness of the baseline methods, we automatically generate $1,929$ adversarial samples targeting the MLP with FGSM. The specification of the generated adversarial dataset is shown in Table I. We train all the methods with the whole normal malware dataset ($12,536$ samples), and test them with all the $1,929$ adversarial samples. In this test, DEEPARMOUR only applies the voting mechanism on the three clssifiers, not using the TFIDF weight or retraining.

Figure 3 presents the performance of all the methods on the adversarial samples. One can see that MLP only gets

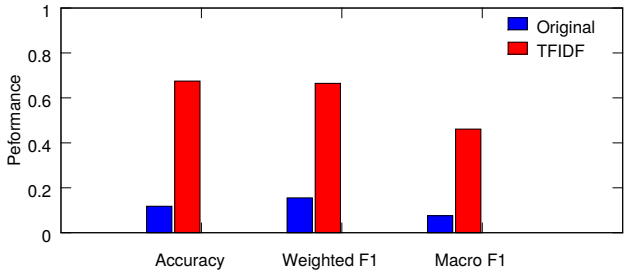

Figure 4. The result of MLP with TFIDF against adversarial samples.

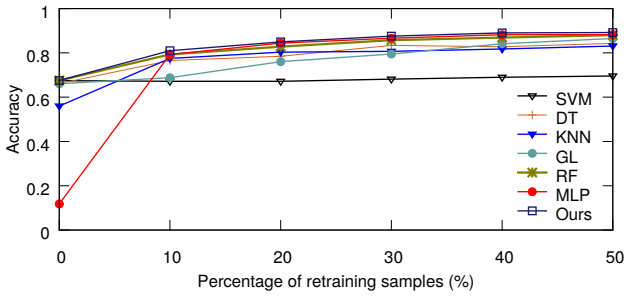

Figure 5. The accuracy of retraining against the adversarial attacks.

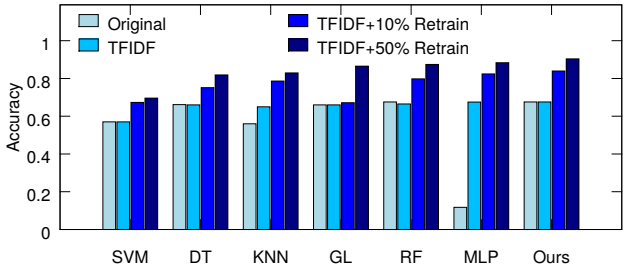

Figure 6. The accuracy of all the methods on the adversarial dataset with the combined defense techniques.

0.1177 accuracy, 0.1551 weighted F1 score, and 0.0754 macro F1 score. Such a low performance shows the effectiveness of the targeted adversarial attack. Also, for all the other basic classifiers, the adversarial samples are able to greatly lower their performance compared with non-adversarial test. Particularly, the accuracy, weighted F1 score, and macro F1 score are below 0.68, 0.65, and 0.27, respectively. The graph learning method, GL, shows the strong robustness against adversarial, since it is able to achieve the highest weighted F1 score 0.65, around 8% higher than the second (DT, 0.57). Although MLP shows low performance, the other two classifiers used in our methods work for different cases. Therefore, voted by the three classifiers, DEEPARMOUR is able to achieve 0.6754, 0.8063, and 0.8063 for accuracy, weighted F1 score, and macro F1 score, respectively.

C. Performance of Different Defense Techniques

We also evaluate the performance of our proposed adversarial defense techniques, including feature reconstruction and adversarial retraining.

Among the feature reconstruction techniques, the graph representation can only be used in the graph learning method which has been proved to be robust against adversarial shown in Figure 3. For the TFIDF weight, we tested on the specific MLP model because it is the one the adversarial attacks are targeting. The performance is significantly increased as shown in Figure 4. Particularly, the accuracy improves from 0.1177 to 0.675, the weighted F1 improves from 0.1551 to 0.6646, and the macro F1 improves from 0.0754 to 0.4613.

Further, we evaluate the effectiveness of retraining against adversarial attacks. Given the trained models on all the non-adversarial samples, we retrain them with a part of the adversarial dataset and test with the rest. Note that we only use the original feature values and do not apply TFIDF weight. We retrain with 10%, 20%, 30%, 40%, and 50% adversarial samples as shown in Figure 5. By only retraining 10% of the adversarial samples, the MLP method is able to significantly improve the accuracy from 0.1177 to 0.7928. DEEPARMOUR is able to achieve 0.81 accuracy by improving 13.5%. The accuracy of all the classifiers improves with the increase of trained adversarial samples, with the exception of SVM dropping a little bit. Particularly, DEEPARMOUR is able to achieve 0.8921 accuracy when retraining with 50% of the adversarial samples.

Figure 6 presents the accuracy of all the methods on the adversarial dataset with the combined defense techniques. Although TFIDF may not increase the accuracy for some methods, its combination with retraining indeed increase the accuracy. For DEEPARMOUR, by using TFIDF weights and retraining 10% adversarial samples, it is able to achieve 0.8394 accuracy. Further, DEEPARMOUR can increase to 0.9036 accuracy by retraining 50% adversarial samples.

D. Parameter Study

In DEEPARMOUR, the voting weights α, β, γ for RF, MLP, and GL, are learned from the training dataset. Observing that the RF and MLP fit better than GL in terms of non-adversarial testing, the weights of RF and MLP will be higher than the GL. In such scenario, the parameters are learned from the normal malware dataset and are set to $0.49, 0.42, 0.09$ for α, β, γ, respectively. These values are used for the test of non-adversarial dataset. For the adversarial attacks, GL is showed to be robust so that its weight will increase accordingly. In the adversarial scenario, the parameters are learned from both the normal malware dataset and our generated adversarial dataset. They are set to $0.3, 0.39, 0.31$ for α, β, γ, respectively. These values are used for all the tests related to adversarial attacks.

VII. RELATED WORK

In this section, we will elaborate the related works towards adversarial countermeasures and malware classification.

Various **adversarial countermeasures** have been explored to protect machine learning algorithms from an adversary. They can be divided into proactive and reactive measures. Proactive measures include techniques such as adversarial retraining [32] [33] and classifier robustifying [34]. Both techniques were adopted in a way in this work. The former countermeasure involves building adversarial samples and retraining your models with these samples. This should improve model accuracy, as well as make it harder for an adversary to craft harmful input. The latter technique, classifier robustifying, generally means building sufficiently deep, or complex models to fit the data better and not be susceptible to adversarial attacks.

Reactive countermeasures include techniques such as adversarial detection [35], input reconstruction [36], and network verification [37]. Adversarial detection involves training special purpose classifiers which typically only decide on weather or not an input is adversarial or not. Input reconstructed, utilized in this work, relies on taking inputs and converting them back to known good features prior to processing them with the machine learning algorithm. Network verification trains models based on various characteristics of the classification models being verified (e.g., per-layer node activation values). In this way, a model can learn when the classification model is performing properly, or improperly, potentially due to an adversarial attack.

Our work is different from three aspects. Firstly, we are focusing on an important security application, malware classification, which is rarely studied against adversarial attack, while most previous works are designed for computer vision related adversarial attacks. Secondly, we design a novel attributed graph with graph learning method for malware API sequence, which provides a new insight against current adversarial attacks. Thirdly, our newly designed weighted voting mechanism uses different machine learning methods on different data representation. It is shown to be robust against potential perturbations brought by adversarial attcks.

Malware classification has been studied for a long time. Most methods use the features extracted from the static and dynamic analysis [38], [39], [40], [41]. The static analysis analyzes the binary code and extracts features such as the hash signature as well as binary code patterns. In contrast, dynamic analysis executes the malware in an isolated environment and monitors its runtime behavior such as system calls, registry operations, and network accesses. Recent works use the deep learning techniques to achieve high accuracy. However, they are proved to be vulnerable to the adversarial as well.

VIII. Discussion

As it is very challenging to build a ML model that is completely resilient to adversary attacks, we sought out to build a model that would make it sufficiently challenging for an adversary to accomplish his goals. Although DeepArmour could be attacked if the majority of classifiers were fooled, we believe achieving this majority is much more challenging than fooling any single model alone. Additionally, as we utilize the TFIDF representation, small perturbations to frequently used features will have less effect on our model accuracy, and thus be less likely to be utilized by an adversary to fool our algorithms. Finally, by retraining our model with our adversarial examples, our model will be more robust to typical ML evasion attacks.

It is possible that our model will still be susceptible to evasion attacks, as the high dimensional space provides a lot of room for samples to move and potentially cross decision boundaries where they should not. However, the methods we outlined in this work we believe should make it sufficiently complex for an adversary to attack the system.

While we have demonstrated that our approach works for malware detection against adversarial attacks, our approach can generalize to other cognitive systems. The two major ideas, feature reconstruction and weighted voting, can be applied to many cases, such as spam detection, intrusion detection, biometric authentication, etc. [10]. Partitcularly, the TFIDF is an efficient measurement for feature importance and has been demonstrated to be effective for many cases, such as, citation [42], object maching in video [43], etc. The attributed graph representation and graph learning technique have been applied to many applications with great performance, such as binary code similarity detection [44], recommender system [45], and drug discovery [46].

IX. Conclusion

In this paper, we have implemented an adversarial attack resilient malware classification system, DeepArmour. DeepArmour applies a voting system with three different malware classification methods, random forest, multi-layer perceptron, and structure2vec. DeepArmour also applies two adversarial defense techniques, feature reconstruction and adversarial retraining. We tested DeepArmour on various scenarios. For non-adversarial scenario, DeepArmour is able to achieve 0.989 accuracy. For unknown adversarial scenario, DeepArmour is able to achieve 0.675 accuracy. Further, by retraining only 10% of the adversarial samples, DeepArmour is able to achieve 0.839 accuracy.

Acknowledgment

This work was supported in part by National Science Foundation CAREER award 1350766 and grants 1618706 and 1717774.

REFERENCES

[1] "Cognitive security white paper, https://cognitivesecuritywhitepaper.mybluemix.net/."

[2] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networksa review," *Pattern recognition*, 2002.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.

[4] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *ACM SIGCOMM Computer Communication Review*, 2014.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv:1312.6199*, 2013.

[6] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv:1206.6389*, 2012.

[7] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *IEEE Security and Privacy*, 2017.

[8] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *arXiv:1511.03034*, 2015.

[9] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *Proceedings of ICLR*, 2015.

[10] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *arXiv:1712.07107*, 2017.

[11] "Deeparmor, endpoint protection, built from api, https://www.sparkcognition.com/product/deeparmor/."

[12] "Cognitive intelligence: Empowering security analysts, defeating polymorphic malware, https://blogs.cisco.com/security/cognitive-intelligence-empowering-security-analysts-defeating-polymorphic-malware."

[13] "Cybereason: Endpoint protection, detection, and response, https://www.cybereason.com/."

[14] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, 2018.

[15] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv:1404.2188*, 2014.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, 2014.

[17] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *Advances in neural information processing systems*, 2016.

[18] G. Ateniese, G. Felici, L. V. Mancini, A. Spognardi, A. Villani, and D. Vitali, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *arXiv:1306.4447*, 2013.

[19] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*. Citeseer, 2009.

[20] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *2014 CCS*. ACM, 2014.

[21] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA*. Springer, 2008.

[22] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML*, 2016.

[23] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *2016 CCS*. ACM, 2016.

[24] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge university press, 2014.

[25] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in computer virology*, 2011.

[26] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of machine learning*. Springer, 2011.

[27] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola, "Feature hashing for large scale multitask learning," *arXiv:0902.2206*, 2009.

[28] Y. Ji, H. Liu, and H. H. Huang, "ispan: Parallel identification of strongly connected components with spanning trees," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 731–742.

[29] P. Kumar and H. H. Huang, "Graphone: A data store for real-time analytics on evolving graphs," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 249–263.

[30] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of SOSP*, 2017.

[31] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv:1611.01236*, 2016.

[32] Y. Wu, D. Bamman, and S. Russell, "Adversarial training for relation extraction," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[33] Y. Dong, H. Su, J. Zhu, and F. Bao, "Towards interpretable deep neural networks by leveraging adversarial examples," *arXiv:1708.05493*, 2017.

[34] J. Bradshaw, A. G. d. G. Matthews, and Z. Ghahramani, "Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks," *arXiv:1707.02476*, 2017.

[35] J. Lu, T. Issaranon, and D. A. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly." in *ICCV*, 2017.

[36] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *2017 CCS*. ACM, 2017.

[37] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017.

[38] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, 2017.

[39] Y. Ji, Q. Li, Y. He, and D. Guo, "Botcatch: leveraging signature and behavior for bot detection," *Security and Communication Networks*, 2015.

[40] Y. Ji, Y. He, X. Jiang, J. Cao, and Q. Li, "Combating the evasion mechanisms of social bots," *computers & security*, 2016.

[41] Y. Ji, Y. He, X. Jiang, and Q. Li, "Towards social botnet behavior detecting in the end host," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2014, pp. 320–327.

[42] K. D. Bollacker, S. Lawrence, and C. L. Giles, "Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications," in *Proceedings of the second international conference on Autonomous agents*, 1998.

[43] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," 2003.

[44] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *2017 CCS*. ACM, 2017.

[45] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv:1706.02263*, 2017.

[46] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.