Attack and Defense of Dynamic Analysis-Based, Adversarial Neural Malware Detection Models

Jack W. Stokes[†]*, De Wang[⋆]*, Mady Marinescu[‡], Marc Marino[‡], and Brian Bussone[‡]
[†]Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA
[⋆]University of Texas at Arlington, Arlington, TX 76019 USA
[‡]Microsoft Corp., One Microsoft Way, Redmond, WA 98052 USA

***Abstract*—Recently researchers have proposed using deep learning-based systems for malware detection. Unfortunately, all deep learning classification systems are vulnerable to adversarial learning-based attacks, or adversarial attacks, where miscreants can avoid detection by the classification algorithm with very few perturbations of the input data. Previous work has studied adversarial attacks against static analysis-based malware classifiers which only classify the content of the unknown file without execution. However, since the majority of malware is either packed or encrypted, malware classification based on static analysis often fails to detect these types of files. To overcome this limitation, anti-malware companies typically perform dynamic analysis by emulating each file in the anti-malware engine or performing in-depth scanning in a virtual machine. These strategies allow the analysis of the malware after unpacking or decryption. In this work, we study different strategies of crafting adversarial samples for dynamic analysis. These strategies operate on sparse, binary inputs in contrast to continuous inputs such as pixels in images. We then study the effects of two, previously proposed defensive mechanisms against crafted adversarial samples including the distillation and ensemble defenses. We also propose and evaluate the weight decay defense. Experiments show that with these three defenses, the number of successfully crafted adversarial samples is reduced compared to an unprotected baseline system. In particular, the ensemble defense is the most resilient to adversarial attacks. Importantly, none of the defenses significantly reduce the classification accuracy for detecting malware. Finally, we show that while adding additional hidden layers to neural models does not significantly improve the malware classification accuracy, it does significantly increase the classifier's robustness to adversarial attacks.**

***Index Terms*—Adversarial Learning, Dynamic Malware Classification**

I. Introduction

As commercial and open source software authors improve the security of their applications, and organizations deploy advanced threat detection systems to harden their defenses, attackers will be forced to employ more sophisticated attacks in order to infect a computer or penetrate an organization's network. One of the primary computer security defenses continues to be commercial anti-malware products. A number of researchers [1], [2], [3], [4], [5], [6] have proposed the use of deep learning for malware classification as a key component of next generation anti-malware systems.

Recently, researchers have also started to study the attacks and defenses of machine learning-based classification systems, and this area is commonly known as adversarial learning. In the remainder of the paper, we refer to the adversarial learning-based attacks as adversarial attacks. In an adversarial attack, miscreants intentionally craft malicious samples which are designed to confuse (*i.e.*, fool) a deployed machine learning model. An adversarial sample is one whose input data is altered in such a way that the perturbation does not change its ground truth label, but the altered sample is misclassified by a trained machine learning model. In some cases such as images [7], the goal is to alter these samples in such a way that they are not perceived by humans to be intentionally corrupted. To be more specific, by perturbing a tiny fraction of the raw input vector features (e.g., pixels) or adding noise with a very small magnitude compared to the original input vector [8], the crafted sample will be misclassified as belonging to a different class. In some cases, the attacker decides to target the mispredicted class to be any desired class. It is a phenomenon that has appeared in some of the deep learning literature [8], [9], but it also exists in shallow linear models [10].

While many authors have focused on adversarial attacks, only a few defenses have been proposed. Goodfellow, *et al.*, [8] proposed training with adversarial samples. In 2015, Papernot, *et al.*, [11] proposed the distillation defense for adversarial learning. More recently, several authors have proposed an ensemble defense [12], [13], [14], [15] for adversarial samples. Xu, *et al.*, [16] proposed a feature squeezing system to detect potential adversarial samples by measuring the difference between the original model and a new model where unnecessary input features have been removed.

Most of the previous research in adversarial learning has typically focused on non-adversarial datasets such as images [7], [8]. Malware classification, on the other hand, is arguably one of the most adversarial environments. To date, relatively few studies have investigated adversarial learning in the field of malware classification. Several papers have focused on the attack side. Hu, *et al.*, [10] study adversarial learning in the context of linear classifiers which are designed to detect malicious PDF (*i.e.*, Adobe Portable Document Format) documents. In [17], Tong, *et al.*, study the effects of iteratively altering malicious PDFs to avoid detection. Hu and Tan [18] propose a generative adversarial network (GAN) for crafting adversarial, malicious Android executable files.

Others have investigated defenses against adversarial attacks for static analysis using the Drebin dataset [19], but no results have been published for dynamic analysis-based malware clas-

* Jack Stokes and De Wang made equal contributions to this work.

sification. Grosse, *et al.*, [20] analyze the distillation defense for a *static* analysis-based, deep malware classification system which only classifies the raw content of the file without execution. Pei, *et al.*, [21] include static analysis of the Drebin dataset when evaluating their whitebox testing system for testing deep learning systems. In [22], Grosse, *et al.*, propose a statistical test for detecting adversarial malware examples.

In this paper, we implement and study several adversarial attacks and defenses for *dynamic* analysis-based, deep learning malware classification systems. All classification models employ deep neural networks (DNNs). We study six different strategies of crafting adversarial malware samples based on the removal of malicious features and the addition of benign features. We evaluate three different defenses for these attacks including the distillation defense and the ensemble defense. We also propose and analyze a new weight decay defense. Results show that the ensemble defense outperforms the other two defenses by a significant margin. Most models yield a similar classification accuracy compared to their baseline systems, which satisfies a key goal of defensive adversarial learning that the defense does not negatively affect the overall detection capability. Finally, while adding additional hidden layers to a neural model only improves the accuracy in a few scenarios, we show that a deep neural network offers much better resilience to adversarial samples compared to its shallow baseline model counterpart. Furthermore, the resilience continues to increase as the number of hidden layers in the DNN increases. A summary of the main contributions of this work includes:

- We are the first to study the efficacy of the distillation defense for dynamic analysis-based, deep malware classification.
- We propose the weight decay defense and analyze its performance in the context of malware classification.
- We demonstrate that the ensemble defense is superior in the context of deep malware classification and are the first to study this defense for either static or dynamic analysis.
- We show that adding additional hidden layers significantly increases the resilience to adversarial attacks.

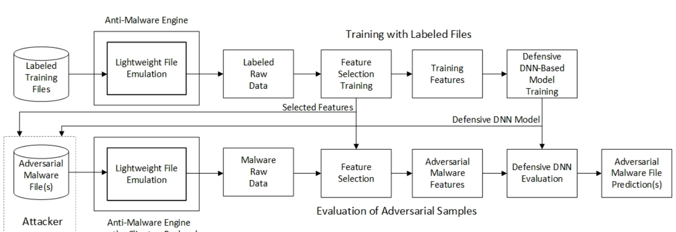


Fig. 1: Overview of the adversarial attack and defense of a dynamic analysis-based malware classification system.

II. Data, System Overview and Threat Model

In this section, we first describe the data that was analyzed in this study. Next we provide a high-level overview of the defender's training and evaluation systems.

Data: The original data for this study was provided to us by the Microsoft's anti-malware team and has been used to train production anti-malware classification systems in the past. The raw data was generated by scanning a large collection of Windows portable executable (PE) files with a production version of the company's anti-malware engine in a virtual machine (VM). The malicious files were collected by the company from security vendor sample exchange programs, product support interactions with customers and anti-virus customers who willing "opt-in" to upload an unknown file encountered on their computer. The benign files were collected from trusted sources such as downloading Adobe Acrobat Reader from the web and copies of commercially available applications. The malware files have the highest confidence level that they are malicious which generally means that they were manually analyzed by one of the company's security analysts.

System Overview: The system overview is depicted in Figure 1. Before an unknown file is executed on the actual operating system, the anti-malware engine first analyzes the file with its lightweight emulator which induces the dynamic behavior of the file. The production engine does not allow network access during emulation to prevent the infection of other computers. The anti-malware engine, that was used to create the raw data for the training, validation and test datasets, generates two sets of logs for each file including unpacked file strings and system API (application protocol interface) calls including their parameters. The first log file that is generated during emulation is a set of unpacked file strings. Typically, a malware file is packed, or encrypted, to make it difficult to reverse engineer by malware analysts. During emulation, text strings, which are included in the PE files, are unpacked and written to the system memory. The emulator's system memory is next scanned to recover null terminated objects which include the original text strings. In addition, the engine also logs the sequence of API calls and their parameters which are generated during execution. The API sequence logs provide an indication of the dynamic behavior of unknown files.

From these two log files, we generate three sets of sparse binary features in the "Feature Selection Training" block to train our deep learning models. We consider each distinct, unpacked file string as a potential feature. Two sets of features are derived from the system call data. First, we generate a potential feature for each distinct value of an API call and input parameter value for a specific input position. Second, we generate all possible combinations of API trigrams (*i.e.*, (k) API call, (k+1) API call, (k+2) API call) as possible features which represent the local behavior of the files.

There are tens of millions of potential features which are generated from the three sets of raw features. Since the neural network cannot process this extremely large set of data, we utilize feature selection using mutual information [23] in order to reduce the final feature set to 50,000 features. Mutual information ranks all of the potential features in terms of which are the most important for classifying whether the file is malicious or benign. If any of these final features are generated during emulation, the corresponding feature will be set to 1 in the sparse, binary input feature for that file. This set of feature vectors is then used to train the deep learning model which has been enhanced to defend against adversarial attacks. Similar steps are used to evaluate an unknown file, which may have been generated by the attacker, to produce the final prediction score.

III. Baseline DNN Malware Classifier

Before discussing the strategies for crafting and defending against adversarial samples, we first review the baseline deep neural network malware classifier used in this study which is illustrated in Figure 2. These parameter settings were chosen by hyperparameter tuning [4]. Even though we have used feature selection to reduce the number of input features to 50,000, this number is still too large to directly input and efficiently train the deep neural network. Therefore we use a sparse random projection matrix [1], [24] to further reduce the input feature dimension from 50,000 to 4,000 for the DNN's input layer. We also tried to reduce the input feature space using principal component analysis, but were not able to generate 4,000 basis vectors. The sparse random projection matrix R is initialized with 1 and -1 as $Pr(R_{i,j} = 1) = Pr(R_{i,j} = -1) = 1/(2\sqrt{d})$, where d is the size of the original input feature vector. All hidden layers have a dimension of 2000. We use the rectified linear unit (ReLU) as the activation function, and dropout [25] is utilized with the dropout rate set to 25% [4]. The maximum number of training epochs is 200, and the minibatch size is set to 250. We use stochastic gradient descent optimization with a momentum term set to 0.9. The initial step size is set to 0.3, but this value is automatically divided in half when the validation loss does not decrease from the previous epoch. All inputs to the DNN are normalized to have zero mean and unit variance. The output layer employs the sigmoid function to generate probabilities for the output predictions.

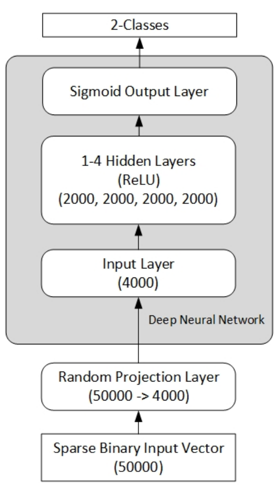


Fig. 2: Model of the baseline deep neural network malware classifier.

IV. Crafting Adversarial Samples

In this section, we describe six iterative methods for crafting adversarial samples. Essentially, the attacker's strategy is to first discover features that have the most influence on the classification output, and then alter their malware to control these features. The Jacobian, which is the forward derivative of the output with respect to the original input, has been proposed in the Jacobian-based saliency map algorithm (JSMA) [11], [26] as a good criterion to help determine these features. For a malware classifier, the prediction output indicates that an unknown file is either malicious or benign. Thus, the attacker's goal is to alter (*i.e.*, perturb) the important features such that the malware classification model incorrectly predicts that a malicious file is benign. To compromise the malware classifier, the attacker can modify their malware to decrease the number of features that are important for a malware prediction, increase the number of features that lead to a benign prediction, or both.

For each iterative attack strategy that simulates an attacker modifying their malware, we alter one feature in the feature vector during each iteration and then re-evaluate the Jacobian with respect to the perturbed sample. We analyze six strategies to craft adversarial samples. The first three methods use the Jacobian information [11], [26] to identify which features to alter:

(1) dec_pos, *i.e.*, disabling the features that would lead the classifier to predict that an unknown file is malware based on the Jacobian of the classification output with respect to the original input features. We define a feature to be a positive feature if the Jacobian with respect to the feature is positive. We call these features *positive features* since they are the key indicators of malware behavior.

(2) inc_neg, *i.e.*, enabling the features that would lead a classifier to predict that an unknown file is benign. These features are called *negative features* with respect to the malware class. A negative feature has a positive Jacobian with respect to the benign class.

(3) dec_pos + inc_neg, *i.e.*, alternatively disabling one positive feature for one iteration and then enabling one negative feature in the next iteration. This strategy investigates whether there is any synergy between removing malicious content and adding benign features in a round robin fashion.

In contrast to the above methods that use the Jacobian information, we also include three, similar "randomized" strategies that do not use the Jacobian for comparison. For these additional algorithms, we randomly select positive features to disable or negative features to enable instead of selecting them using the rank of the Jacobian's forward derivatives. Thus, the additional strategies include: (4) randomized dec_pos, (5) randomized inc_neg_random, and (6) randomized dec_pos + inc_neg.

V. Defensive Methods

In this section, we review three methods for defending against adversarial attacks including the distillation, ensemble, and weight decay defenses. Although the distillation and ensemble defenses have been previously proposed, the weight decay defense is new. Only the distillation defense has been previously explored to defend against adversarial attacks in malware detection applications, and that work was done in the context of static malware classification [20].

Distillation Defense: The first defense we study is the distillation defense [11], [20] where the model is trained using knowledge distillation. Knowledge distillation is typically used to distill the knowledge learned from a large model into a smaller network making the smaller model more efficient in terms of its memory, energy, or processing time for deployment. However, in adversarial learning, the goal is to make the distilled model more robust to adversarial perturbations, instead of focusing on compressing the network size.

The motivation of using model distillation as a defense mechanism is that with a higher temperature during the distillation process, the error surface of the learned model can be smoothed. We denote the function learned by the neural network model as F. During the inference stage, the feature vector is input into the trained network and transformed into logit scores $z \in \mathbb{R}^{c \times 1}$. Then a softmax function is used to convert those scores into probabilities with respect to each class. Mathematically, the Jacobian's forward derivative of the output with respect to the input can be calculated as follows [11], [26], [27]. For notational clarity, we denote the denominator of the softmax function as $h(x) = \sum_{k=1}^{c} (exp(z_k)/T)$, where T is the temperature used during distillation. Thus, we have:

$$\frac{\partial F_i}{\partial x_j} = \frac{\partial}{\partial x_j}(\frac{e^{z_i/T}}{h(x)}) = \frac{1}{T}\frac{e^{z_i/T}}{h^2(x)}(\sum_{k=1}^{c}(\frac{\partial z_i}{\partial x_j} - \frac{\partial z_k}{\partial x_j})e^{z_k/T}). \quad (1)$$

From (1), we see that as the derivative becomes smaller with higher temperature, the model is less sensitive to adversarial perturbations.

Ensemble Defense: The ensemble defense for extraction attacks and evasion attacks has been recently proposed by several authors [12], [13], [15] for tree ensemble classifiers. In this work, we study the ensemble defense with neural networks. The idea behind the ensemble defense is intuitive. It may be easy for an attacker to craft adversarial samples to compromise an individual detection model, but it is much more difficult for them to create samples which fool a set of models in an ensemble with different properties. We employ a "majority vote" ensemble defense in this work. We first train an ensemble with E classifiers where E is an odd number. During prediction, an unknown file is predicted to be malware if the majority (*i.e.*, $> E/2$) of the classifiers predict that the file is malicious.

Weight Decay Defense: The third defense we propose and study is the weight decay defense. Weight decay is typically used to prevent overfitting of machine learning models. The ℓ_2 norm of a weight matrix is defined as the square sum of all the elements. By adding an ℓ_2 penalty of the model weights in the objective function during optimization, the model is encouraged to prefer smaller magnitude weights since large values are penalized by the objective function.

With a smaller magnitude of weights, the function parameterized by the neural network is smoother, and therefore, changes in the input space lead to smaller changes in the output of a deep learning model. We conjecture that weight decay could help alleviate the vulnerability of a deep learning system against adversarial attacks.

VI. Experimental Results

In this section, we evaluate the adversarial defenses against the different attack strategies described in the previous sections.

Data Preparation and Setup: In some cases, multiple files can share the same input feature vector. Therefore, we only include the first instance of a file with a unique input feature vector and discard any remaining duplicates. After deduplicating the data from 4.5 million unique files, we have input data and labels from 2,373,671 files including 43.8% malware and 56.2% benign. A file is assigned the label of 1 if it is malware and 0 if it is benign. We then randomly split the original dataset into a training set, validation set, and test set including 1,523,978, 268,937, and 580,756 files, respectively.

In our training, we implement all models using the Microsoft Cognitive Toolkit (CNTK) [28]. All models are derived from the baseline model described in Section III. We use the Adam optimizer [29] for training where the initial step size is set to 0.1. Training proceeds for each step size until no further improvement is observed in the validation error. At that point, CNTK halves the step size for subsequent epochs. We train for a maximum of 200 epochs, but CNTK implements early stopping when no additional improvement in the validation error is observed for a minimum step size of 1e-4.

Baseline Classifier: Before investigating the various defenses, we first analyze the performance of the baseline malware classifier by measuring the test error rates in Table I for a range of DNN hidden layers, H, varying from 1 to 4. This table reports both the test error rates for all files with distinct cyptographic hash values and files with distinct (*i.e.*, deduplicated) feature vectors, which are described above. We note that the test error rates increase by a factor close to three for all values of H for the dataset with distinct feature vectors. Next, we provide the receiver operating characteristic (ROC) curves in Figure 3 for the baseline classifier trained and tested with the distinct dataset. If we had measured these results using the standard method of individual files based on a cryptographic hash, the ROC curves would appear to be even more effective.

Malware classifiers need to operate at very low false positive rates to avoid false positive detections which may result in the removal of critical operating system and legitimate application files. Thus, our desired operating point is a false positive rate (FPR) of 0.01%. While the DNNs with multiple hidden layers offer equivalent performance at higher false positive rates compared to a shallow neural network with one hidden layer, the figure indicates that the DNNs offer improved performance at very low false positive rates. In particular, the false positive rate of the shallow neural model immediately jumps to over 0.015% which is above our desired operating point.

Distillation Defense: We next analyze the performance of the distillation defense system for all malware and benign files. The ROC curves of the DNN systems employing the distillation defense are presented in Figure 4 for $T = 10$ for a range of hidden layers. We make several observations from this figure. All models provide multiple operating points below FPR = 0.01% which allows better fine-tuning. Also, we obtain

Number Hidden Layers	Test Error Rate (%) for All Files	Test Error Rate (%) for Files with Distinct Feature Vectors
1	0.43455	1.1378272
2	0.4368	1.2053255
3	0.4603	1.1762255
4	0.4824	1.1619338

TABLE I: Test error rates of the baseline malware classifier for different numbers of hidden layers.

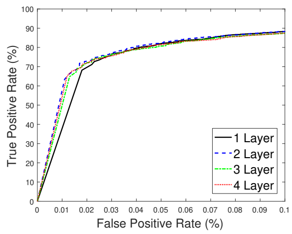


Fig. 3: ROC curves of the baseline malware classifier for different numbers of hidden layers.

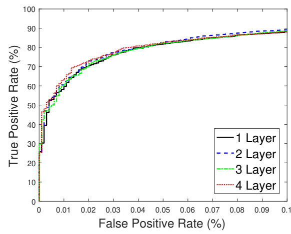


Fig. 4: ROC curves of the malware classifiers with the distillation defense with $T = 10$ for different numbers of hidden layers.

a small lift in the performance for the DNN with 4 hidden layers for $T = 10$. All models offers similar performance above an FPR = 0.02% compared to the baseline classifiers in Figure 3.

In Figure 5, we next investigate the effectiveness of the six adversarial sample crafting strategies for the baseline classifiier and distillation defense, with temperatures $T \in \{2, 10\}$, for model depths $H \in \{1, 2, 3, 4\}$. In each iteration, a single feature is modified, and the generated sample is evaluated by the trained model to test whether the sample is misclassified. From Figure 5, we make several observations. Generally, the distilled models follow a similar trend with regard to the six strategies for crafting adversarial samples, where dec_pos and dec_pos+inc_neg are the two most effective strategies for the attacker. With a higher distillation temperature, it becomes much harder to craft adversarial samples for the distilled model. If the same number of features is perturbed, the success rate for crafting adversarial samples is reduced significantly for models distilled with a higher temperature. This result is because the error surface of the distilled model is smoothed for higher temperatures, such that the output is less sensitive with respect to the input.

We summarize the success of the different iterative strategies for crafting adversarial samples after iteration 20 in Figures 6 for the baseline classifier and Figure 7 for $T = 10$. The figures indicate that shallow networks with $H = 1$ hidden layers are the most susceptible to successfully crafted adversarial samples. We see that using the Jacobian information can help to craft more adversarial samples with the same number of perturbed features than its randomized counterparts. From the attacker's perspective, the dec_pos strategy (switching off positive malware features) is the most effective approach for crafting adversarial samples for the full defense with $T = 10$. Likewise, dec_pos + inc_neg (alternatively switching off a positive feature and switching on a negative feature) is more effective than inc_neg (switching on negative features). This is fortunate from the defender's perspective because it requires the attacker to potentially spend more effort implementing alternative strategies for removing malicious features.

Weight Decay Defense: We next present an analysis of the proposed weight decay defense. We trained the malware classification models using different strengths of weight decay regularization, $D \in \{0.0001, 0.0005, 0.001, 0.01\}$, and plot the ROC curves for $D \in \{0.0001, 0.0005\}$ in Figures 8-9, respectively. In general, the true positive rates drop with increasing values of D. For $D \in \{0.001, 0.01\}$, the models' ROC curves were not acceptable.

We summarize all combinations of the weight decay strength and hidden layer depth in terms of defense to adversarial attacks in Figures 10 and 11 for $D \in \{0.0001, 0.0005\}$, respectively. The best overall resilience of this model defense to the six adversarial sample crafting strategies for iteration 20 also employs $D = 0.0001$ and is summarized in Figure 10. For comparison, we also summarize the defensive capabilities for $D = 0.0005$ in Figure 11. Figure 10 shows that the resilience to adversarial sample crafting strategies also increases as the hidden layer depth increases.

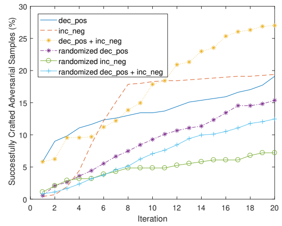
Ensemble Defense: Finally, we present the results for the ensemble defense on our dataset. In Figure 13, we show the ROC curves for an ensemble with $E = 5$ classifiers. Ensembles with other numbers of base classifiers (*e.g.*, 3, 7) offer similar results.

The summary results after 20 iterations for $E = 3$ and $E = 5$ classifiers are shown in Figure 14 and Figure 15, respectively. The figures indicate that increasing the number of classifiers in the ensemble increases the difficulty of successfully crafting adversarial examples. Furthermore, the ensemble defense greatly reduces the percentage of successfully crafted samples compared to the results for the baseline classifier in Figure 6, the distillation defense with $T = 10$ in Figure 7, and the weight decay defense in Figures 10 and 11.

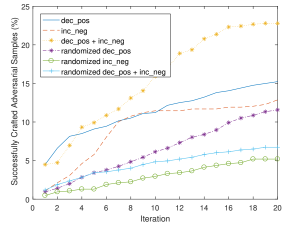
VII. Related Work

Adversarial Attacks: Goodfellow, *et al.*, [8] demonstrated that deep learning models can be fooled by crafting adversarial samples from the original input data by adding a perturbation on the direction of the sign of the model's cost function gradient. This method is known as the *Fast Gradient Sign* method. Papernot, *et al.*, [7] proposed the Jacobian-based Saliency Map Algorithm (JSMA) method, based on model distillation, to craft adversarial attack samples on black box models which we investigate in this paper. The authors in [7] found that adversarial samples are transferable among models, *i.e.*, the adversarial samples crafted for one model can also mislead the classification of other models.

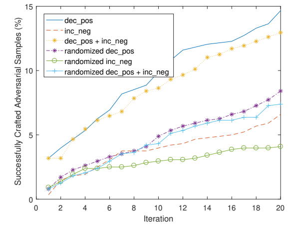
The distillation defense that is studied in the paper is proposed by Papernot, *et al.* in [11]. Carlini and Wagner [30] developed a new type of adversarial attack based on the L_2 regularization of an optimization function and showed that distillation is not effective against this attack for images.



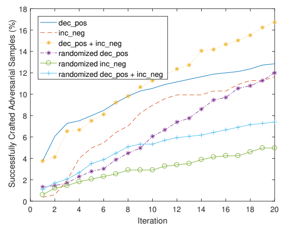
(a) $H = 1$, Baseline



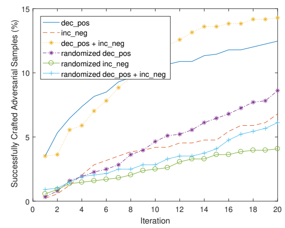
(b) $H = 1$, $T = 2$



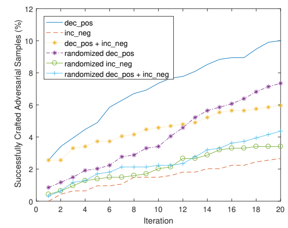
(c) $H = 1$, $T = 10$



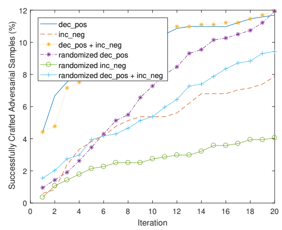
(d) $H = 2$, Baseline



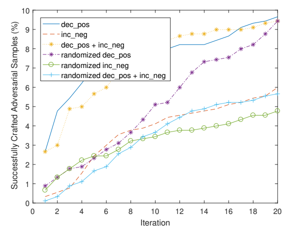
(e) $H = 2$, $T = 2$



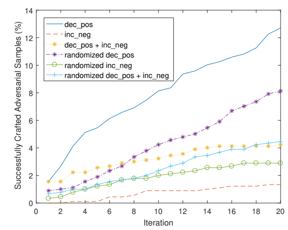
(f) $H = 2$, $T = 10$



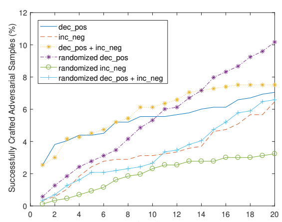
(g) $H = 3$, Baseline



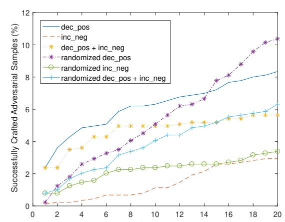
(h) $H = 3$, $T = 2$



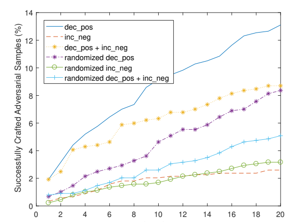
(i) $H = 3$, $T = 10$



(j) $H = 4$, Baseline



(k) $H = 4$, $T = 2$



(l) $H = 4$, $T = 10$

Fig. 5: Success rates of adversarial samples against the baseline classier and classifiers using the defensive distillation with temperatures, $T \in 2, 10$. Each subfigure shows the results of a DNN with a different number of hidden layers, H.

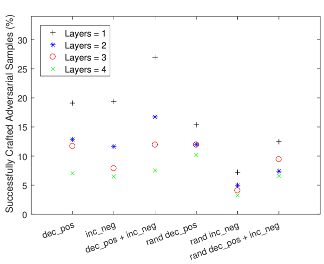


Fig. 6: Percentage of successfully crafted adversarial samples for different sample crafting strategies for the baseline model with no defense.

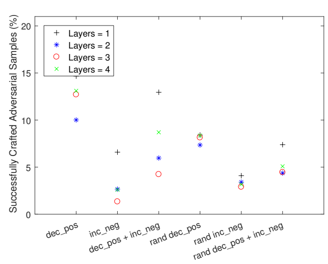


Fig. 7: Percentage of successfully crafted adversarial samples for different sample crafting strategies with the distillation defense and $T = 10$.

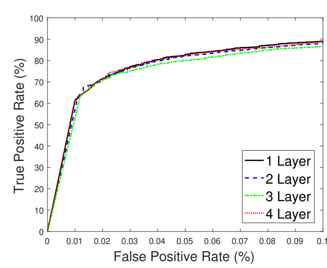


Fig. 8: ROC curves of the malware classifiers for the weight decay defense with $D = 0.0001$ for different numbers of hidden layers.

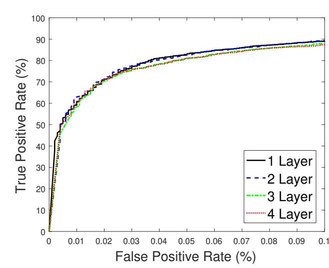


Fig. 9: ROC curves of the malware classifiers for the weight decay defense with $D = 0.0005$ for different numbers of hidden layers.

Several authors [12], [13], [14], [15] have proposed using an ensemble of models to avoid different type of adversarial attacks. For example, the authors in [13] proposed using an ensemble of models to improve the privacy of deployed models since attackers will only be able to obtain an approximation

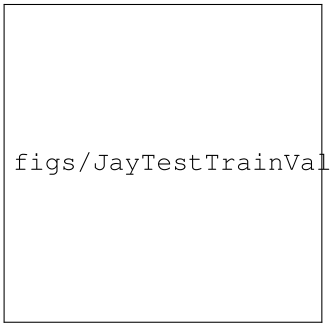


Fig. 10: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with the weight decay defense and $D = 0.0001$.

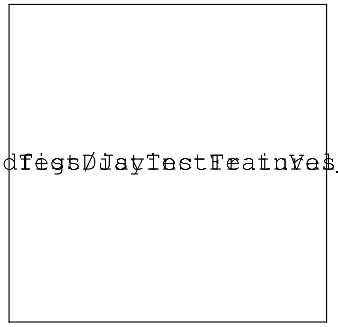


Fig. 11: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with the weight decay defense and $D = 0.0005$.

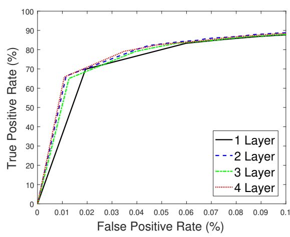


Fig. 12: ROC curves of the ensemble malware classifier with $E = 3$ classifiers for different numbers of hidden layers.

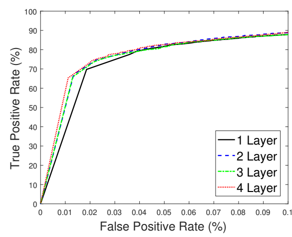


Fig. 13: ROC curves of the ensemble malware classifier with $E = 5$ classifiers for different numbers of hidden layers.

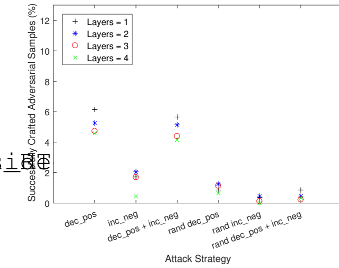


Fig. 14: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with the $E = 3$ ensemble defense.

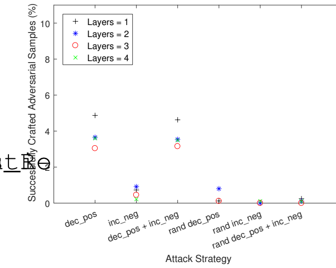


Fig. 15: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with the $E = 5$ ensemble defense.

of the target prediction function. Kantchelian, *et al.*, [12] proposed two algorithms for evasion attacks on tree ensemble classifiers, like gradient boosted trees and random forests. However, each tree classifier is very weak compared with a full-fledged neural network.

Malware Classification: Several deep learning malware classifiers are proposed in [1], [2], [3], [4], [5], [6]. The first study of deep learning for a DNN malware classifier was presented in [1]. Similar to our results, the authors found that a shallow neural network slightly outperformed a DNN on dynamic analysis-based malware classification. Saxe, *et al.*, studied DNNs in the context of static malware classification in [2]. Huang and Stokes proposed a deep, multi-task approach for dynamic analysis which simultaneously tries to optimize predicting a) if a file is malicious or benign and b) the file's family if it is malware or returning a benign label in the case it is clean. In [3], the authors propose a two-stage approach where the first stage employs a language-model, using a recurrent neural network (RNN) or an echo state network (ESN), to first learn an embedding of the behavior of the file based on its system call events. This embedding then serves as the features for a DNN in the second stage. Athiwaratkun, *et al.*, [5] explored similar architectures for deep malware classification using long short-term memory (LSTM) or a gated recurrent units (GRU) for the language model, as well as a separate architecture using a character-level convoluation neural network (CNN). In [6], Kolosnjaji, *et al.*, propose an alternative model also employing a CNN and an LSTM.

Several authors have proposed methods for creating adversarial malware samples. In [10], Xu, *et al.*, propose a system which uses a genetic algorithm to generate adversarial samples which can be mispredicted by a classifier. The system assumes access to the classifier's output score. The authors demonstrate that their system can automatically create 500 malicious PDF files that are classified as benign by the PDFrate [31] and Hidost [32] systems.

Hu and Tan [18] propose a generative adversarial network (GAN) to create adversarial malware samples. In their work, the authors assume that the attackers know the features which are employed by the malware classifier, but they do not know the classification model or its parameters. They use static analysis where the features are API calls and a sparse binary feature is constructed to indicate which APIs were called by the program. Furthermore, the authors assume that the prediction score from the model is reported from the malware classification model.

Grosse, *et al.*, [20] study the distillation defense for *static* analysis-based malware classification. Similar to this paper, the authors assume that the attacker has access to all of the deep learning malware classifier's model parameter. In our work, we also consider the distillation defense for *dynamic* analysis-based malware classification. In addition, we evaluate the ensemble defense for a dynamic malware classifier. In [21], the authors also evaluate adversarial learning-based attacks on the Drebin malware database. In another recent paper, Grosse, *et al.*, [22] add a separate class for adversarial samples and propose a statistical hypothesis test to identify adversarial samples.

VIII. Conclusion

In this paper, we investigated six different adversarial learning attack strategies against a dynamic analysis-based, deep learning malware classification system. We analyzed the effectiveness of two previously proposed defensive methods including the distillation defense and ensemble defense. Although

Grosse, *et al.*, [20] have investigated the distillation defense for *static* analysis, it is important to evaluate this method for dynamic analysis datasets. While distillation helped with static analysis, it was not that effective for our dynamic analysis dataset, which is more representative of how malware classification is done in practice. Furthermore, this is the first work to analyze the effectiveness of the ensemble defense for malware classification in general for either static or dynamic analysis. We also proposed and analyzed the weight decay defense.

All three defenses offer comparable classification accuracies compared to a standard deep learning baseline system. Thus, they achieve a key goal in adversarial learning of not significantly reducing the accuracy compared to a system without any adversarial learning defenses. In addition, *deep* learning models offer better resilience to adversarial attacks than the shallow baseline models in all cases.

Results show that the ensemble classifier provides significantly better resilience against adversarial attacks for this dataset when compared to the other defenses, but requires more computational resources for both training and inference. The distillation defense offers the second best resistance. It helps to reduce the effectiveness of removing important malicious features, but its effectiveness is limited.

Most importantly, none of these defenses are completely effective in preventing adversarial attacks for dynamic analysis-based malware classification systems. While reasonably effective in the case of perturbing randomly selected features, even the ensemble defense only makes it more time consuming for an attacker who uses the Jacobian-based attack strategy to craft adversarial malware samples. Researchers keep creating better attacks, but we really need the research community to propose better defenses to protect our computer systems.

References

[1] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 3422–3426.

[2] J. Saxe and K. Berlin, "Deep neural network based malware detection using two-dimensional binary program features," *Malware Conference (MALCON)*, 2015.

[3] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1916–1920.

[4] W. Huang and J. W. Stokes, "Mtnet: A multi-task neural network for dynamic malware classfication," in *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016, pp. 399–418.

[5] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2482–2486.

[6] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer International Publishing, 2016, pp. 137–149.

[7] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2017.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Proceedings of the International Conference on Learning Representations (ICML)*, 2015.

[9] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 427–436.

[10] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[11] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[12] A. Kantchelian, J.D.Tygar, and A. D.Joseph, "Evasion and hardening of tree ensemble classifiers," *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[13] F. Tramer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," *Proceedings of the USENIX Security Symposium*, 2016.

[14] J. Feng, T. Zahavy, B. Kang, H. Xu, and S. Mannor, "Ensemble robustness of deep learning algorithms," *arXiv preprint arXiv:1602.02389v3*, 2016.

[15] F. Tramer, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[16] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[17] L. Tong, B. Li, C. Hajaj, C. Xiao, and Y. Vorobeychik, "A framework for validating models of evasion attacks on machine learning, with application to pdf malware detection," *arXiv preprint arXiv:1708.08327v3*, 2017.

[18] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint 1702.05983*, 2017.

[19] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

[20] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2017.

[21] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Symposium on Operating Systems Principles (SOSP)*, October 2017, pp. 1–18.

[22] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," in *arXiv preprint arXiv:1702.06280v2*, 2017.

[23] C. D. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval*. Cambridge University Press, 2009.

[24] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ICDM)*, 2006, pp. 287–296.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313

[26] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2015.

[27] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Neural Information Processing Systems (NIPS) Deep Learning Workshop*, 2014.

[28] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2135–2135.

[29] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the International Conference for Learning Representations (ICLR)*, 2015.

[30] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 39–57.

[31] C. Smutz and A. Stavrou, "Malicious pdf detection using metadata and structural features," *Technical report*, 2012.

[32] N. Srndic and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2013.