Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8876632, 11 pages
https://doi.org/10.1155/2020/8876632

Research Article

BMOP: Bidirectional Universal Adversarial Learning for Binary OpCode Features

Xiang Li[1], **Yuanping Nie**[1], **Zhi Wang**[2], **Xiaohui Kuang**[1], **Kefan Qiu**[2], **Cheng Qian**[1] and **Gang Zhao**[1]

[1]*National Key Laboratory of Science and Technology on Information System Security, Beijing 100101, China*
[2]*Nankai University, Tianjin, China*

Correspondence should be addressed to Yuanping Nie; yuanpingnie@nudt.edu.cn and Zhi Wang; zwang@nankai.edu.cn

Received 25 June 2020; Revised 30 August 2020; Accepted 28 October 2020; Published 3 December 2020

Academic Editor: Weizhi Meng

For malware detection, current state-of-the-art research concentrates on machine learning techniques. Binary *n*-gram OpCode features are commonly used for malicious code identification and classification with high accuracy. Binary OpCode modification is much more difficult than modification of image pixels. Traditional adversarial perturbation methods could not be applied on OpCode directly. In this paper, we propose a bidirectional universal adversarial learning method for effective binary OpCode perturbation from both benign and malicious perspectives. Benign features are those OpCodes that represent benign behaviours, while malicious features are OpCodes for malicious behaviours. From a large dataset of benign and malicious binary applications, we select the most significant benign and malicious OpCode features based on the feature SHAP value in the trained machine learning model. We implement an OpCode modification method that insert benign OpCodes into executables as garbage codes without execution and modify malicious OpCodes by equivalent replacement preserving execution semantics. The experimental results show that the *benign and malicious OpCode perturbation* (BMOP) method could bypass malicious code detection models based on the SVM, XGBoost, and DNN algorithms.

1. Introduction

With the vigorous development of the information industry, security incidents caused by malware are also in an inexhaustible variety. The "2018-2019 Annual Security Report" issued by the world-renowned antivirus testing agency AV-TEST pointed out that nearly 400,000 new malwares appear everyday, so computer protection software has to resist more than 3.9 malicious programs per second.

As the malware constantly evolve rapidly, antivirus software also continues improving. In recent years, the state-of-the-art machine learning techniques have gradually been applied in the malware detection and classification, which could effectively handle a huge number of malware samples and achieve fairly good detection results. Schultz et al. [1] utilized machine learning algorithms to detect malicious code which achieved 97.76% accuracy. Michael et al. [2] exploited the situation of system state changes to detect malicious code behaviours, which reached 91% detection rate on a malware dataset containing more than 4,000 samples. Abou-Assaleh et al. [3] extracted the *n*-gram binary character features from malicious samples and achieved 98% accuracy. In the malware binary code, there are some binary OpCode sequences that are more significant as compared to benign programs which could be used as feature points for machine learning. Currently, the *n*-gram OpCode features have been commonly used by machine learning-based detection models. The *n*-gram OpCode features have much less computational overhead compared to dynamic features, such as API call sequences. Moreover, the *n*-gram OpCode features could cover much more code area than dynamic features which are limited by the virtual machine execution environment.

At present, the robustness of malware detection models is getting more and more attention. The adversarial machine learning techniques are widely used to test the robustness of

machine learning models in the fields of image recognition and speech recognition [4–6], also in the computer security field such as spam filtering. Adversarial machine learning could effectively find out a malicious input data perturbation to attack or cause a malfunction to the target machine learning models. However, traditional adversarial perturbation methods could not be applied on binary OpCodes directly. The binary OpCode features are sustainable that the binary OpCode modification is much difficult with program execution and semantic preserving.

In this paper, we propose BMOP, a bidirectional universal adversarial learning method for effective binary OpCode perturbation from both benign and malicious perspectives. The benign features are those OpCodes that significantly represent benign behaviours, while malicious features are OpCodes dominate malicious behaviours. From a large dataset of benign and malicious binary applications, we select the most important benign and malicious OpCode features based on the feature SHAP values calculated from the trained machine learning models. We implement a binary OpCode modification platform which could insert adversarial benign OpCodes into application as garbage codes without execution and modify adversarial malicious OpCodes by equivalent replacement preserving code semantics. We test BMOP performance on three standard machine learning models: SVM, XGBoost, and DNN. The experimental results show that the BMOP method could completely fool the malware detection models.

In a nutshell, we make the following contributions:

(i) We propose BMOP, a universal adversarial learning method to assess the malware detection models based on OpCode features. The BMOP leverages SHAP algorithm to find out the significant malware-oriented features and the goodware-oriented benign features. Under the guidance of significant features founded by the SHAP algorithm, BMOP could effectively modify malware OpCode sequence in two opposite directions: (1) enlarge or add the OpCodes related to the significant goodware features and (2) weaken or delete the OpCode sequences related to the malware

(ii) We build an OpCode modification platform to change binary executable OpCode features. After modification, the functionality of new binary executable is preserved which is consistent with the original samples

(iii) We evaluate BMOP on a dataset encompassing 11,997 binary executables. The experiment results show that BMOP is effective to craft new adversarial samples to break through malware detection models, respectively, using SVM, XGBoost, and DNN algorithms

The rest of the paper is organized as follows: Section 2 reviews related work, Section 3 presents the details of BMOP method, the evaluation experiments and discussions are presented in Section 4, while Section 5 concludes the paper.

2. Related Work

An array of works focused on malware detection using various machine learning algorithms. Majorities of them can be considered classification-type solutions. Anderson et al. propose a malware detection method which utilizes the instruction traces [7]. They modify a malware analysis framework called Ether to collect the instruction traces and then create the similarity matrix according to the graph kernel combinations between instruction trace graphs. The matrix is finally fed into SVM to perform classification by using 2-gram-based Markov chain to estimate the transition probability. Two different similarity evaluation methods are applied to construct the matrix, namely, Gaussian kernel and spectral kernel, which can measure the local similarity and global similarity between graphs. Santos et al. combine dynamic and static features to propose a hybrid malware detector, which uses static analysis to model binary files into OpCode sequences for feature extraction, and dynamic analysis is used to monitor the operations, system calls, and exceptions meanwhile [8]. Saxe and Berlin take byte entropy histogram, PE import information and PE metadata as features, and use deep neural network and Bayesian calibration model to detect malwares [9]. Hardy et al. also apply the deep learning framework and combine SAE model to do the malware detection based on Windows API call features [10]. Raff et al. optimize the selected features for detecting malwares. Firstly, they only select the n-gram sequences that have high frequency (more than 1%), and a coarse-grain selection method is applied to reduce the data amount. The final features are determined after the logistic regression test in lasso and resilience models [11]. Given the argument that was based on n-gram have giant computation overhead while the effect is limited, Raff et al. propose that the source of feature extraction can be limited to the binary file headers, and then, the n-gram features can be directly obtained from the raw byte stream. They use fully associative neural network and regression network as the classifier in this work [12]. To avoid the problem that feature extraction may impede the learning process, Raff et al. present a work that feed the whole binary files into the convolution neural network, and the neural network do the feature extraction and classification directly. The CNN can convert the embedded byte stream into features so that more feature information can be obtained [13]. Xu et al. point that while the graph matching-based algorithm is widely used in similarity evaluation of multiple platform binary file, it is quite time consuming and not highly accurate. They propose a neural network-based evaluation method called Gemini. Gemini can convert the graph into feature vector in its graph-embedded network layer and evaluate the difference between feature vectors [14]. Xu et al. notice that one fixed feature of malware is that they are destined to change the control flow and data structure; therefore, they propose a machine learning method based on virtual memory access pattern. A large amount of memory access information is processed to form a histogram so that significant features can be preserved and distinguished. Several classification methods are applied in this work, including logistic regression, SVM, and random forest [15].

Current adversarial learning targeted malware technology focus on two aspects: attack during training phase and attack during prediction phase. The former mainly refers to poison attack, which tries to modify the statistical features of a dataset, so that the machine learning model can be compromised. In most cases, the primary data is encrypted to prevent being modified easily; however, in real-world scenario, the dataset may vary as the environment changes, and correspondingly the machine learning model should be retrained, which leaves opportunity for attackers to operate the training data. Attack during the prediction phase generally means to take use of some weaknesses in a machine learning model. Biggio et al. propose an optimized further prioritized label flipping (FPFL) attack. This method modifies the train data and random hyperplane that is far from the decision boundary of SVM, which can incur lower accuracy compared with original FPFL attack [16]. Hu et al. states that although the current machine learning-based antivirus software has a blackbox structure, the features that it checks can still be tested and confirmed. This work proposes MalGAN, which can generate an adversarial sample to pass though the blackbox check model [17]. Kreuk et al. present a GAN model for malware detector which uses raw binary files as input. This model can generate one-key representation of adversarial discrete byte steam to reconstitute binary file. The reconstitute binary file can avoid being detected while keeping the original capability [18]. Tram et al. use FGSM to efficiently produce adversarial samples, which attach noise to raw image in the direction of gradient descent [4]. Sarkar et al. propose two blackbox attacks: UPSET and ANGRI. For a machine learning model that classify samples into N sets,

UPSET tries to produce K image-independent, universal disturbance. When attached with the disturbance, the image in fact does not belong to the original category while the machine learning model still classifies it to the same category. In the contrary, ANGRI produces image-dependent, specific disturbance for each unique image [6]. Carlini et al. propose C&W attack, which generates adversarial samples using L0-norm, Euclidean distance, and Chebyshev distance. C&W attack has faster generation speed and great portability, which means the generated samples can also be used in the blackbox attack [19].

SHAP (SHapley Additive exPlanations) is a unified framework for interpreting predictions which was proposed by Li et al. [26]. SHAP assigns each feature an importance value for a particular prediction. Recently, several researches connect the explanation of machine learning with adversarial learning [23–25]. Fidel et al. utilize SHAP values which computed for the internal layers of a DNN, to detect whether the input image is normal or adversarial [25]. Coull et al. utilize SHAP values to interpret a byte-based deep neural network for malware classification [24]. Their study shows that the DNN does not learn why malware is malicious, but only finds the most significant difference between malware and goodware through feature statistic. Such statistical difference can be exploited by hackers to evade detection. Giorgio uses SHAP to guide the feature selection for implementing a clean-label poisoning attack [23].

In this paper, we not only use SHAP to find out the significant malware-oriented features but also the goodware-oriented features. Under the guidance of significant features founded by the SHAP algorithm, the BMOP is able to modify malware OpCode sequence in two opposite directions: (1) enlarge or add the OpCodes related to the significant goodware features and (2) weaken or delete the OpCode sequences related to the malware. After the malware OpCode modification, the functionality of new binary executable is preserved which is consistent with the original malware sample.

3. Methods

3.1. Overview. In this section, we present a bidirectional universal adversarial learning method (BMOP) on representing and discovering the important features, which influence the machine learning model classification ability on malware detection domain. The method has four components:

(1) Malware representation: in this component, we firstly represent the malware with OpCode and employ the n-gram method to extract the features from the malwares. Since the n-gram method will generate massive and redundant features, we choose the TF-IDF approach to select the most valuable features to represent the malwares

(2) Model training and explanation: in this component, we first train a well-tuned XGBoost model which can effectively classify the malwares and goodwares. We use the SHAP model to calculate the importance of each feature. Note that our model can calculate the importance score of positive and negative features

(3) Feature selection: we use the importance score to choose the malicious and benign features of the malware

(4) Adversarial example generation: according to the malicious and benign features, we use the equivalent instruct replace method to reduce the impact of malicious features and insert garbage codes to increase the impact of benign features. Our generation method will not break the integrities and functionalities of the malwares. The overview of our method is shown in Figure 1

3.2. Malware Representation

3.2.1. Feature Representation. In this paper, we extract OpCode features from PE samples and express the samples with OpCode expressions. The OpCode is the part of instructions that are specified in the machine instruction language to perform the operation. A complete machine language instruction generally contains an operation code and several operands. Depending on the CPU architecture, the operands for OpCode operations may include registers, values in memory, values stored in the stack, I/O ports, and buses. The operation of the operation code can include arithmetic, data operation, logic operation, and program control. In the past

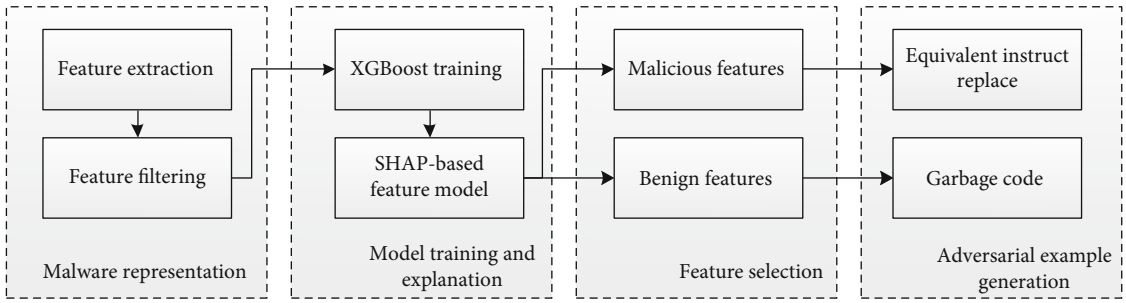

Figure 1: Overview of the BMOP method.

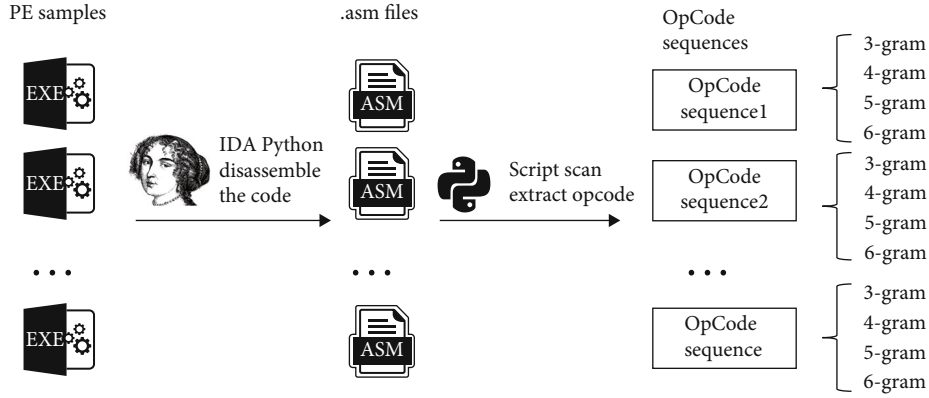

Figure 2: The process of the OpCode generation.

decade, some malware detection research based on binary information of files has been started gradually. The paper [20] shows that according to the statistical analysis of OpCodes, there are obvious differences between malicious code and normal software in the distribution of OpCodes. Malicious code often uses some rare OpCodes, and a method of malicious code detection based on OpCodes is proposed. Based on the distribution of OpCodes, Shahzad and Lavesson [21] employed detection method based on the n-gram feature. N-gram is a string with all substrings of length n. A string is simply divided into fixed length n substrings. In 2015, Microsoft launched a malicious code classification competition on Kaggle (https://www.kaggle.com/), and the champion team from the University of Pittsburgh also used the OpCode n-gram feature.

The problem of locating malicious code signature can be transformed into the problem of finding malicious OpCode n-gram sequence in samples, because from the perspective of compilation principle, binary machine code, and assembly instruction's OpCode can be transformed into each other. From the perspective of malicious code classification, there are different families of malicious codes, and the malicious codes of the same family often have similar functions. Although the malicious code authors use various polymorphic deformation techniques to avoid killing, they do not modify the function of the program, mainly because the external view of the program changes, and they still have a high degree of similarity in the local sequence of operation codes, in which a similar part of it can be seen as the "fingerprint" or "gene" of this malicious code family.

3.2.2. Feature Filtering. To extract OpCodes from PE samples, we need to disassemble the samples. Disassembly translates the machine instructions stored in the PE file into a language that is more easily readable by human beings, that is, assembly instructions. Finally, the sequence of OpCodes generated in the disassembly process is extracted, and its logical order is the same as that of the operation codes appearing in the executable file, without considering other information (such as memory location and register). In this paper, we use IDA to realize disassembly. We transform IDA Python script to disassemble the PE sample automatically. We generate ASM file to store assembly instructions and traverse the ASM file to obtain OpCode sequence. Finally, the corresponding OpCode n-gram is generated according to different N values. The process is shown in Figure 2.

We collected ASM files generated by disassembly from the open sources and divided the training set and test set according to the ratio of 7 : 3. And, the 2-gram, 3-gram, and 4-gram sequences of operation codes are extracted from each ASM file. In the field of machine learning, a large number of features will not only increase the training time of the model but also sometimes cannot improve the accuracy of the model, or even reduce the accuracy. Therefore, we need to select features and reduce the number of features used in training while maintaining the accuracy of the model. We filter features according to document frequency (DF) of OpCode n-gram and calculate term frequency inverse document frequency (TF-IDF) weight for each n-gram. Formula (1) gives the calculation method of word frequency. TF-IDF combines the word frequency (TF) of an entry in a document

with the frequency (DF) of the entry in the document set, and its calculation method is shown in formula (2), where n is the total number of documents in the whole document set, and DF is the number of documents containing the entry.

$$\mathrm{TF} = \frac{\text{term frequency}}{\max\ (\text{term frequency in document})}, \tag{1}$$

$$\text{TF-IDF} = \mathrm{TF} * \log\left(\frac{N}{DF}\right). \tag{2}$$

In order to reduce the number of OpCode n-grams, we first select the first 1000 OpCode n-grams according to the document frequency and calculate their TF-IDF weights as features for model training.

3.3. Model Training and Explanation

3.3.1. Machine Learning Model Selection and Training. Moskovitch et al. [22] deployed four classification algorithms based on OpCode n-gram: artificial neural network (ANN), decision tree (DT), naive Bayes (NB), and promotion decision tree (BDT). Through comparative experiments, it is proved that the BDT has the best performance in this task. In this paper, we evaluate deep neural network (DNN), support vector machine (SVM), and XGBoost in our dataset. The evaluation results are shown at Experiments. The XGBoost model has better performance on detecting the malware. Note that the XGBoost model's features have better interpretability. Therefore, we use the XGBoost model to distinguish between malicious samples and benign samples.

When training the XGBoost model, there are four important parameters: eta, max_depth, num_round, and min_child_weight. The four parameters are very important to the training results and fitting degree of the XGBoost model. In order to train a high-performance model, we need to select the influence of different parameters on the model. The parameter eta is the learning rate of XGBoost. The number of eta will influence the overfitting and less fitting of the model. In this paper, we set eta as 0.05. The max_depth decides every decision tree's max depth, which also impacts the fitting degree of the XGBoost model. The max_depth is set to 7 in this work. Num_round is the number of training iterations. When the loss of the model is small enough after a certain iteration, the training process will be terminated. The min_child_weight is the sum of minimum leaf node weight. We set the value of this parameter as 1.

3.3.2. SHAP Feature Selection Model. After obtaining the XGBoost model, we need to explain the prediction results of the model and locate the important features that affect the decision making of the model. The machine learning model can find the difference between malicious code and normal software. We analyze the features that lead to input samples being classified as malicious tags by the model and use these OpCode n-gram malicious features to realize intelligent derivation of malicious code signatures. Feature importance is a traditional method to explain machine learning model decision. In this paper, we list three methods to measure the importance of different types of features: Weight is the total number of times feature f splits in all XGBoost subtrees. Cover is the average coverage of feature f to input samples when it splits in all XGBoost subtrees. Gain is the average value of feature f improving the accuracy of the model at each split. The results of these three types of feature importance calculation are inconsistent, which is not conducive to our accurate evaluation of the important features of the model. Therefore, we cannot analyze the relationship between the features and the prediction results of the XGBoost model according to the importance of features, nor can we interpret the positive and negative effects of different features on the prediction results of samples.

In order to solve the inconsistency of feature importance in XGBoost, random forest, and other tree set models and to explain the influence of feature on prediction results, we use the SHAP framework based on game theory. The SHAP framework can calculate the SHAP values for all features of the test samples, which can reflect the specific impact of each feature on the prediction results. As for the malicious code detection model, a test sample s has the feature. If it is calculated that the SHAP value of the feature f in the sample s is positive, it means that the feature f classifies the sample s as malicious tag 1; if the SHAP value is negative, it means that the feature f classifies the sample s as benign tag 0. The results are shown in Figures 3 and 4.

Figures 3 and 4 are a scatter diagram. Each row in the figure represents a feature. The x-axis abscissa is the corresponding snap value of the feature, and the y-axis ordinate is the feature name. Each point represents a sample in the training set, and the color of the point represents the value of the feature corresponding to the y-axis. The redder the color is, the larger the value of the feature and the bluer is the value of the feature. Because the XGBoost model in this paper uses 1000 n-gram features, the y-axis of Figure 5 cannot display all feature names. Figure 3 controls the number of features to 30 and visualizes the distribution of their SHAP values again.

According to the distribution of the SHAP values of the features, we can locate the important features intuitively and get the correlation between the features and the prediction results. As shown in Figure 4, the 4-gram "mov + and + or + mov" feature will significantly affect the prediction results. The red dots are basically concentrated in the area where the swap value is greater than zero, and the blue dots are basically concentrated in the area where the swap value is less than zero. It can be seen that the increase of the weight of this feature will increase the probability that the samples are predicted as malicious tags by the model. The "mov + and + or + mov" feature with large weight is a typical malicious feature. The "div+mulps" and "ror+ucomiss" is typical benign features, for the red dots are basically concentrated in the area where the swap value is less than zero, and the blues are basically concentrated in the area where the swap value is greater than zero.

3.3.3. Adversarial Example Generation. For malicious features, we use instruction substitution to blur the malicious

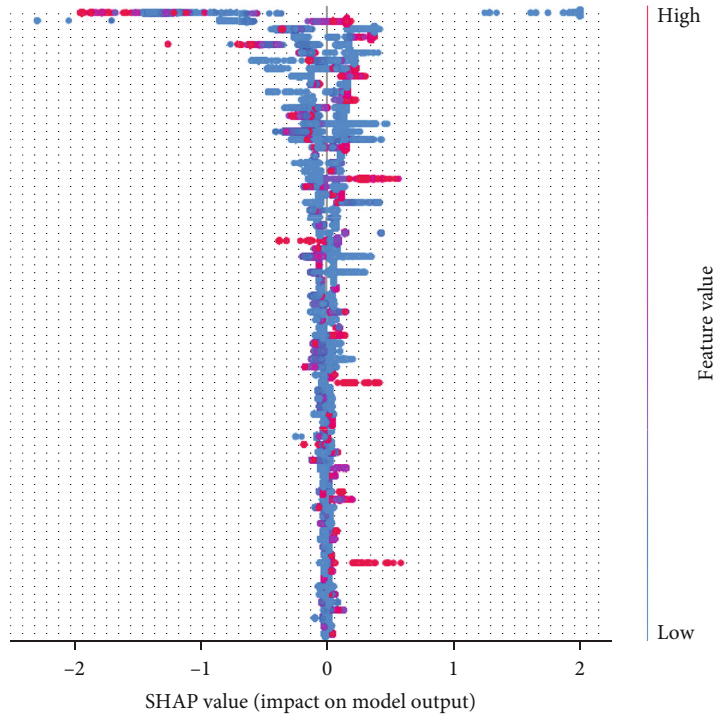

Figure 3: Distribution of all the characteristic SHAP values.

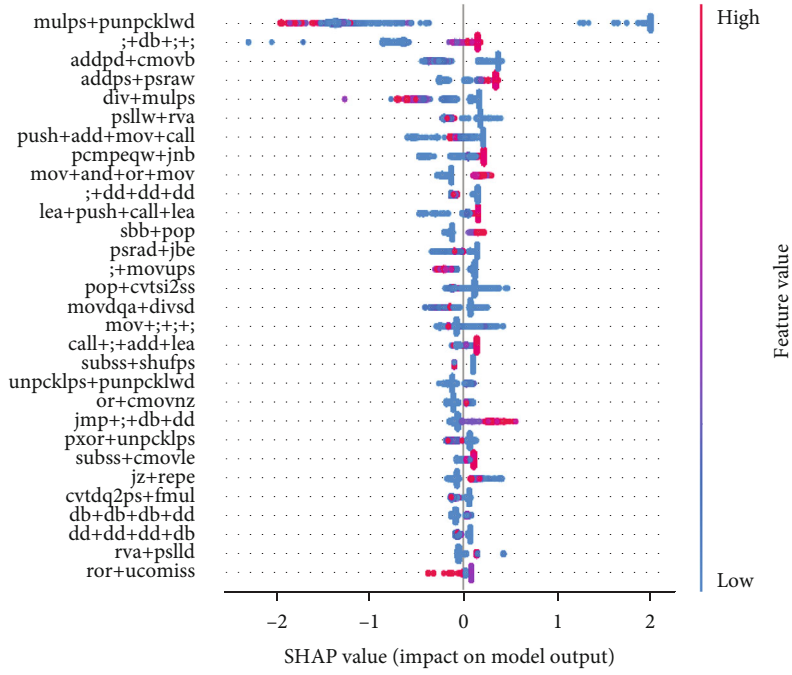

Figure 4: Distribution of some of the characteristic SHAP values.

features. Instruction substitution is using equivalent instruction sequences to replace original instruction sequences in a program. For example, the instruction "mov eax, ebx" can be replaced by "push ebx; pop eax". Correspondingly, the OpCode sequence "mov + and + or + mov" mentioned above can be replaced with "push + pop + and + or + push + pop." There are a variety of instructions in X86 or ARM instruction sets so that it provides sufficient conditions for the implementation of instruction substitution. In addition, when replacing the instruction, it is also necessary to consider two situations caused by the different length of the replacement instructions and the original instructions: instruction contraction and instruction expansion. Instruction contraction means that the length of the new instructions after

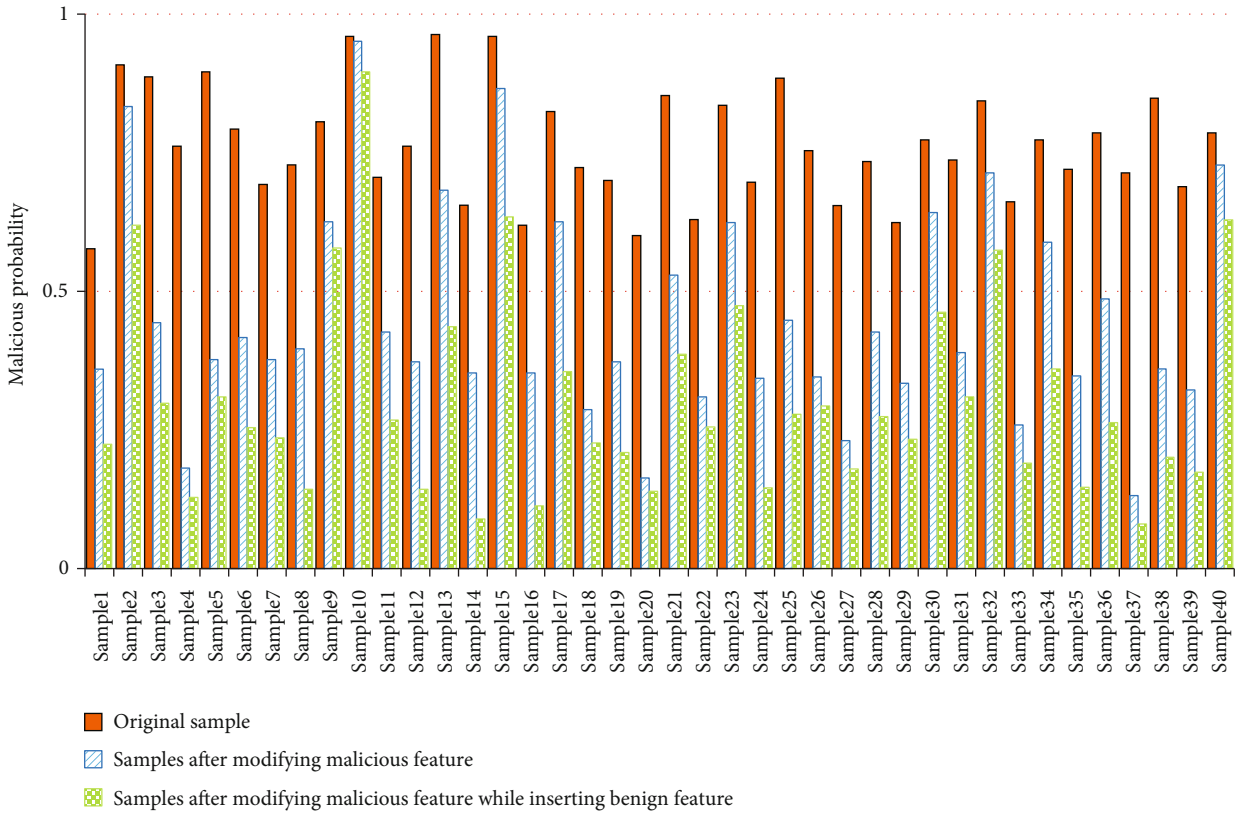

Figure 5: Comparison of XGBoost model results before and after adversarial perturbation.

replacement is less than the original instructions. In view of this situation, we use the method of inserting "nop" instruction to fill the vacant part. Instruction expansion means that the length of the new instructions after replacement is greater than the original instructions. Because this situation is more complicated to modify, a little negligence will destroy the integrity of the program. Therefore, only short or equal length instructions are used in instruction substitution in this paper, thus avoiding the problem of instruction expansion.

For benign features, we employ the instruct injection method. The method firstly builds a benign feature database according to the SHAP method. And then, search the target OpCode with the continuous zero zone. Finally, we calculate the length of the continuous zero zone and insert binary sequences from the benign feature database randomly until the continuous zero zone is filled by the binary sequences. Therefore, the malicious code functionality is not broke.

4. Experiments

In this paper, we organize two kinds of experiment. In the first experiment, we train three malware detection models using SVM, XGBoost, and DNN algorithms for the purpose of choosing the best model for feature extraction. In the second experiment, we test BMOP performance on three standard machine learning models: SVM, XGBoost, and DNN.

Table 1: Dataset of machine learning comparison.

Dataset	Train dataset	Test dataset	Total
Malware	6018	2950	8968
Goodware	1909	1120	3029
Total	7927	4070	11997

Table 2: Performance of three malware detection models based on SVM, DNN, and XGBoost algorithms.

Feature	SVM	DNN	XGBoost
Precision@Malware	0.94	0.95	0.99
Recall@Malware	0.92	0.95	0.99
Precision@Goodware	0.95	0.96	0.96
Recall@Goodware	0.96	0.97	0.99

4.1. Experiment A

4.1.1. Dataset. Dataset is a prerequisite for training models. In order to improve the authenticity and typicality of the malicious code, we select VirusShare3 as the data source. http://VirusShare.com is a malicious code sample library. By continuously releasing the latest captured malicious code, it provides malicious code samples for security research, incident response, and judicial forensics. Currently, more than 34 million malicious code samples have been collected. Since 80% of the malicious code has been packed, which affects the accuracy of feature extraction, we selected the malicious code

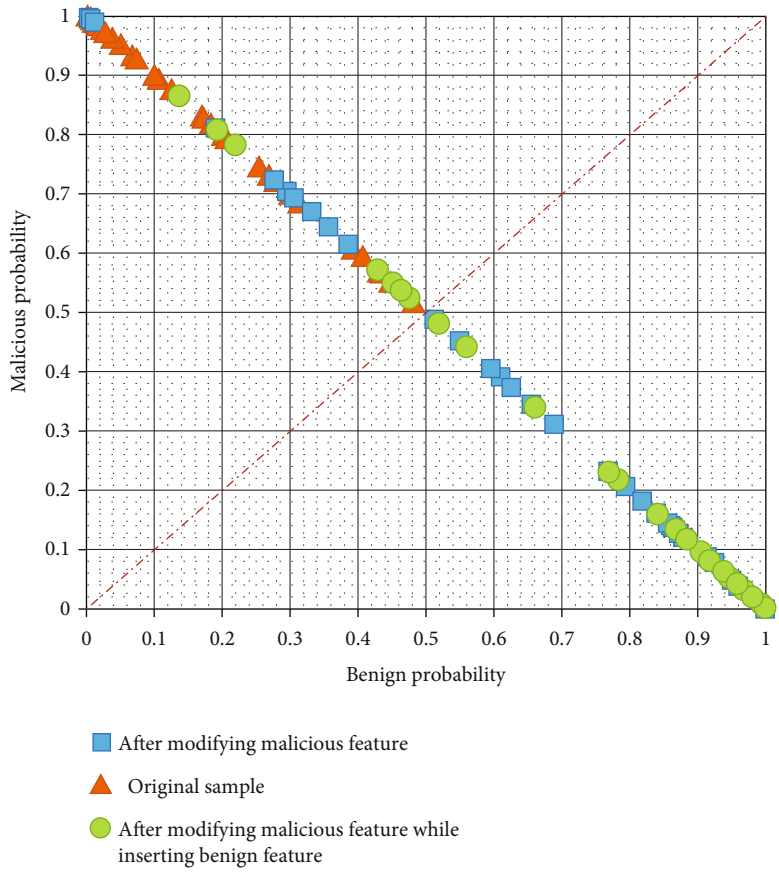

Figure 6: Comparison of DNN model results before and after adversarial perturbation.

samples in VirusShare that have been unpacked by the CodexGigas team. So we can not only ensure the authenticity of the samples but also eliminate the unpacking preprocessing step and accelerate the extraction of features from the samples.

In addition, we collected windows software that has undergone 360 security tests, and PE files on Window7 and Window10 as a benign dataset. The dataset used for training and testing is shown in Table 1.

4.1.2. Results. In this experiment, we evaluate three classic machine learning models with the dataset mentioned before. We use precision and recall to measure the performance of each model. The result is shown at Table 2. As we can see, all the machine learning models perform well on classifying malware and goodware. The precision and recall rates with malware and goodware are higher than 92%. The DNN model perform better than the SVM model. Specifically, the XGBoost model has the best performance on malware classification. That is the reason that we use the XGBoost model in this paper.

4.2. Experiment B

4.2.1. Experimentation. In the adversarial experiment, we randomly select 20 samples that were detected as malware by our malware detection models in experiment A. We first modify the adversarial malicious OpCodes of the 20 samples by equivalent replacement and test the samples' malicious probability on three machine learning models: SVM, XGBoost, and DNN. We trained in experiment A. Then, we continuously modify the 20 samples by inserting adversarial benign OpCodes and also test the samples' malicious probability on three machine learning models above.

5. Result and Discussion

Figure 5 shows the malicious probability of the malware samples predicted by the XGBoost model before and after modification, where the y-axis ordinate is the malicious probability of the samples. The orange solid color histogram in the left represents the malicious probability of original malware samples before modification. The blue histogram with twill in the middle represents the malicious probability of the samples only modifying the malicious features. And the green histogram in the right represents the malicious probability of the samples both modifying the malicious features and inserting the benign features. It can be seen that the malicious probability of the original samples are all above 0.5, indicating that these 20 original samples are all predicted as malicious samples by the XGBoost model. After modifying the malicious feature, only 6 samples' malicious probabilities are above 0.5. That means the success rate is about 70%. While after inserting the benign features, the number of the

malicious samples predicted by the XGBoost model as malware fell to 4. That means the success rate is about 80%.

Figure 6 shows the probability distribution of the malware samples predicted by the DNN model before and after modification, where the x-axis abscissa is benign probability and the y-axis is malicious probability. Orange triangle represents the original malicious samples before modification. Blue square represents the samples only modifying the malicious features, and yellow circle represents the samples modifying the malicious features while inserting the benign features. It can be seen from the experiment results that the original samples are all distributed in the left half of the diagonal, indicating that these 20 original samples are all detected as malicious samples by the DNN model. After modifying the malicious features, there are 15 samples are distributed in the right half of the diagonal. That means the success rate is about 75%. After modifying the malicious features while inserting the benign features, there are 16 samples are distributed in the right half of the diagonal. The distribution of all samples tends to move in the benign direction.

Table 3 shows the distance from the hyperplane of the 20 samples in the SVM model before and after modification. The distances in column 2 are all positive. That means the 20 original samples are all detected as malicious in the SVM model. In column 3, only 5 samples' distances are positive, set in italic. That means after modifying the malicious features, 75% of the samples are detected as benign in the SVM model. In column 4, only 3 samples' distances are positive, set in italic. That means after inserting benign features, 85% of the samples can fool the SVM malware detection model.

Further, we evaluate the performance of our BMOP method among the three models on three perspectives: the performance of only modifying the malicious features, the performance of inserting benign features based on modifying the malicious features, and the performance of modifying malicious features while inserting benign features.

For the convenience of comparison, we first normalize the results of the SVM model, and the normalization method is shown in formula (3).

$$\text{new\_value} = \frac{\text{value} - \min\_\text{value}}{\max\_\text{value} - \min\_\text{value}}. \quad (3)$$

Tables 4–6 show the performance of only modifying the malicious features, only inserting the benign features, and both modifying malicious features and inserting benign features on XGBoost, DNN, and SVM. We calculate the maximum, minimum, and average rate of changes. The result shows that the method of modifying the malicious features is more effective to the DNN model; the method of both modifying the malicious features and inserting benign features is also more effective to the DNN model.

It can be seen from the above experimental results that our BMOP method can not only effectively fool the XGBoost model for feature extraction but also fool other malicious code detection models such as SVM and DNN, and ever more effective to DNN than XGBoost. The method of inserting benign features can effectively increase the benign

Table 3: Comparison of SVM model results before and after adversarial perturbation.

Samples	Original samples	Modifying malicious features	Modifying malicious features while inserting benign features
Sample 1	0.05086315	-2.18679011	-2.43210622
Sample 2	3.20902217	*2.44127492*	*1.73218008*
Sample 3	0.95373816	-1.03511355	-1.95118186
Sample 4	0.97534874	-3.88666444	-4.18261757
Sample 5	0.96937306	-0.71789673	-1.23858554
Sample 6	0.99823124	*0.20839891*	-0.33175236
Sample 7	0.86933176	-0.55050156	-0.90512465
Sample 8	1.73947293	-1.75670475	-2.53824632
Sample 9	0.92906255	-2.95616363	-3.64117659
Sample 10	1.73611227	*1.5037305*	*1.16266138*
Sample 11	0.91294243	-1.49449477	-1.87826411
Sample 12	1.15371952	-1.12874049	-2.27395693
Sample 13	1.64748286	*0.45159495*	*0.13265951*
Sample 14	0.45969358	-2.17869359	-3.49242179
Sample 15	1.30165517	*0.15682057*	-0.18701269
Sample 16	0.74412254	-2.16283582	-3.11282724
Sample 17	0.54949236	-0.24393718	-0.69554975
Sample 18	0.73796295	-1.44053385	-1.8794114
Sample 19	0.64732315	-1.73397271	-2.50161645
Sample 20	0.46883346	-2.24840841	-2.67792805
Sample 21	3.42704634	*1.86171241*	*0.61453235*
Sample 22	0.81871514	-0.27838477	-0.83195452
Sample 23	0.88682458	*0.18604343*	-0.94792017
Sample 24	0.65501845	-0.93248342	-2.36510321
Sample 25	2.23824429	*0.13697118*	-0.63235912
Sample 26	0.58418869	-1.28744881	-1.83461894
Sample 27	3.20013715	*1.41282654*	*0.11346283*
Sample 28	2.38352685	-0.92597681	-1.48730921
Sample 29	1.00946153	-1.38958651	-2.23940657
Sample 30	1.09366996	-0.47387622	-1.21498436
Sample 31	2.19822903	-0.65426643	-1.30946112
Sample 32	1.91556312	*1.41146936*	*0.14379123*
Sample 33	2.86444982	*1.15772314*	*0.13346381*
Sample 34	1.42083990	-0.80189551	-1.46218423
Sample 35	3.45755239	*2.55865362*	*1.56219303*
Sample 36	1.17483269	-1.09462018	-2.34512719
Sample 37	0.36242719	-2.18791299	-2.65438128
Sample 38	0.60150393	-1.14826383	-1.85323934
Sample 39	1.13201853	-1.00857442	-2.03467805
Sample 40	3.18099898	*0.82366729*	-0.45012989

Table 4: Comparison the performance of only modifying malicious features.

Performance (malicious)	XGBoost	DNN	SVM
Max	57.94%	99.54%	65.78%
Min	0.65%	0.18%	3.14%
Average	26.54%	51.18%	26.80%

Table 5: Comparison the performance of inserting benign features after modifying malicious features.

Performance (benign)	XGBoost	DNN	SVM
Max	29.36%	38.54%	19.38%
Min	2.39%	0.00%	3.32%
Average	19.45%	13.50%	10.27%

Table 6: Comparison the performance of both modifying malicious features and inserting benign features.

Performance (malicious & benign)	XGBoost	DNN	SVM
Max	63.20%	99.54%	69.78%
Min	6.32%	13.50%	7.76%
Average	44.71%	64.69%	37.08%

probability of the malware although the benign features would not be executed. In addition, the above experimental results also reflect from the side that the features used by the malicious code detection model based on SVM and DNN may be similar to XGBoost in predicting malicious code.

6. Conclusions

We presented the BMOP, a universal method for assessing the robustness of malware detection models against OpCode perturbation. Our work details the selection of adversarial OpCode features from both benign and malicious perspectives and the crafting process of adversarial binary samples with functionality preserving. We evaluated the BMOP method on a huge array of malware samples. The experiment results show that BMOP is effective and efficient to locate the most significant benign and malicious OpCode sequences from 11,997 binary samples and craft adversarial executables by increase benign OpCodes and replace malicious OpCodes to bypass malware detection models which use SVM, XGBoost, or DNN algorithms.

Data Availability

The data used to support the findings of this study have been deposited in http://www.github.com/NKQiuKF/BMOP

Disclosure

The present work is an extension of our DSC2020 submission [26]. The main addition is introducing the benign perspective instead of only malicious perspective to achieve a bidirectional universal adversarial learning framework.

Conflicts of Interest

The author(s) declare(s) that they have no conflicts of interest.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant 61872202, Natural Science Foundation of Tianjin under Grant 19JCYBJC15500, and 2019 Tianjin New Generation AI Technology Key Project (19ZXZNGX00090).

References

[1] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pp. 38–49, Oakland, CA, USA, 2001.

[2] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 178–197, Gold Goast, QLD, Australia, 2007.

[3] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based detection of new malicious code," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004*, pp. 41-42, Hong Kong, China, 2004.

[4] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: attacks and defenses," 2017, https://arxiv.org/abs/1705.07204.

[5] S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, Honolulu, HI, USA, 2017.

[6] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa, "UPSET and ANGRI: breaking high performance image classifiers," 2017, https://arxiv.org/abs/1707.01159.

[7] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in Computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[8] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: a static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions*, pp. 271–280, Springer, 2013.

[9] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 11–20, Fajardo, Puerto Rico, 2015.

[10] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: a deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN)*, p. 61, Las Wegas, Nevada, USA, 2016, The Steering Committee of The World Congress in Computer

Science, Computer Engineering and Applied Computing (WorldComp).

[11] E. Raff, R. Zak, R. Cox et al., "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, 2018.

[12] E. Raff, J. Sylvester, and C. Nicholas, "Learning the pe header, malware detection with minimal domain knowledge," in *AISec '17: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 121–132, Dallas Texas USA, 2017.

[13] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA, 2018.

[14] 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 *CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 363–376, Dallas TX, USA, 2017.

[15] Z. Xu, S. Ray, P. Subramanyan, and S. Malik, "Malware detection using machine learning based analysis of virtual memory access patterns," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 169–174, Lausanne, Switzerland, 2017.

[16] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in *Proceedings of the Asian Conference on Machine Learning, PMLR*, pp. 97–112, Taoyuan, Taiwain, 2011.

[17] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, https://arxiv.org/abs/1702.05983.

[18] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Adversarial examples on discrete sequences for beating whole-binary malware detection," 2018, https://128.84.21.199/abs/1802.04528v1.

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

[20] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–156, 2007.

[21] R. K. Shahzad and N. Lavesson, "Detecting scareware by mining variable length instruction sequences," in *2011 Information Security for South Africa*, Johannesburg, South Africa, 2011.

[22] R. Moskovitch, C. Feher, N. Tzachar et al., "Unknown malcode detection using OPCODE representation," in *Intelligence and Security Informatics*, pp. 204–215, Springer, 2008.

[23] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Exploring backdoor poisoning attacks against malware classifiers," 2020, https://arxiv.org/abs/2003.01031.

[24] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," in *2019 IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, 2019.

[25] G. Fidel, R. Bitton, and A. Shabtai, "When explainability meets adversarial learning: detecting adversarial examples using shap signatures," 2019, https://arxiv.org/abs/1909.03418.

[26] X. Li, K. Qiu, C. Qian, and G. Zhao, "An adversarial machine learning method based on OpCode N-grams feature in malware detection," in *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, pp. 380–387, Hong Kong, 2020.