Research Article

Binary Black-Box Adversarial Attacks with Evolutionary Learning against IoT Malware Detection

Fangwei Wang ^{1,2}, Yuanyuan Lu ¹, Changguang Wang ^{1,2} and Qingru Li ^{1,2}

¹College of Computer & Cyber Security, Hebei Normal University, Shijiazhuang 050024, China

²Key Lab of Network & Information Security of Hebei Province, Shijiazhuang 050024, China

Correspondence should be addressed to Changguang Wang; wangcg@hebtu.edu.cn and Qingru Li; liqingru2006@163.com

Received 28 May 2021; Accepted 12 August 2021; Published 30 August 2021

Academic Editor: Dr. Muhammad Shafiq

Copyright © 2021 Fangwei Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

5G is about to open Pandora's box of security threats to the Internet of Things (IoT). Key technologies, such as network function virtualization and edge computing introduced by the 5G network, bring new security threats and risks to the Internet infrastructure. Therefore, higher detection and defense against malware are required. Nowadays, deep learning (DL) is widely used in malware detection. Recently, research has demonstrated that adversarial attacks have posed a hazard to DL-based models. The key issue of enhancing the antiattack performance of malware detection systems that are used to detect adversarial attacks is to generate effective adversarial samples. However, numerous existing methods to generate adversarial samples are manual feature extraction or using white-box models, which makes it not applicable in the actual scenarios. This paper presents an effective binary manipulation-based attack framework, which generates adversarial samples with an evolutionary learning algorithm. The framework chooses some appropriate action sequences to modify malicious samples. Thus, the modified malware can successfully circumvent the detection system. The evolutionary algorithm can adaptively simplify the modification actions and make the adversarial sample more targeted. Our approach can efficiently generate adversarial samples without human intervention. The generated adversarial samples can effectively combat DL-based malware detection models while preserving the consistency of the executable and malicious behavior of the original malware samples. We apply the generated adversarial samples to attack the detection engines of VirusTotal. Experimental results illustrate that the adversarial samples generated by our method reach an evasion success rate of 47.8%, which outperforms other attack methods. By adding adversarial samples in the training process, the MalConv network is retrained. We show that the detection accuracy is improved by 10.3%.

1. Introduction

With the commercialization and popularization of 5G, the IoT is coming closer to reality [1]. Meanwhile, with the scale expansion of connected terminals, data storage, and utilization, security issues are becoming more and more complex. As the methods of network crime are also constantly updated, the probability of network attack is greatly increased, which is not conducive to protecting personal privacy [2]. Therefore, in the 5G era, designing a model with good robustness is an important issue.

At present, malware attacks remain as one of the most urgent security issues users facing. In the last decade, deep neural network-based malware detection has fulfilled

remarkable achievements [3]. A growing community of researchers is attempting to apply deep learning to malware detection and classification tasks [4–9]. Saxe and Berlin [10] extracted the binary features of PE files, which are portable executable ones under Windows operation systems and utilized a four-layer feed-forward neural network to detect malware. Kalash et al. [11] transformed malware binaries into greyscale images and classified malware by the use of a CNN. The DL-based malicious detection and classification models are now widely used.

However, much recent work indicates that adversarial attacks can cause serious damage to deep neural networks [12–15]. Adversarial examples in computer vision applications have been widely proven. In malware detection,

adversarial modifications often need minor changes to malicious binaries. Different from language and images, codes are discrete sequences, which means that the generation technique of adversarial samples in images cannot be transferred to the malware detection field. Moreover, a minor change in the sequence may result in its functionality be changed completely. For example, in a binary file, changing a single byte may lead to a completely ineffective bytecode or distinct functionality. Therefore, it remains a great challenge to implement practical black-box attacks on malware binary-based deep learning models. Recently, a series of research works have been done in adversarial attacks. Hu and Tan [16] proposed adversarial modification of feature vectors. However, the malware binaries were not modified actually. This method cannot guarantee that the modified feature vector can be converted to actual binaries. Moreover, it destroyed the format or affected the functionality of the malware. Anderson et al. [17] directly modified malware binaries to perform adversarial attacks. In theory, these methods cannot disrupt the original functionality of the malware. However, in practice, we have found that these seemingly reliable methods also damage the malicious functionality. During an adversarial attack, if the malicious functionality of the original sample is destroyed, the adversarial attack is invalid. Therefore, it is necessary to perform malicious functionality detection on generated adversarial samples. However, most of the previous work did not address this issue. Besides, some of the previous work was done in a white-box adversarial model [18–20]. The white-box adversarial model requires knowing malware classifier architecture, making their methods impractical in real network environments. Therefore, while retaining the primary malicious functionality of the binaries, it is a great challenge to implement a practical black-box attack on the malware detection model based on deep learning.

This paper proposes an evolutionary algorithm-based adversarial sample generation method. In our approach, the generated samples by rewriting the file structure and adding adversarial information evade successfully the malware detection model, while preserving the original behavior of PE files. We test 1000 PE samples in four popular antivirus software on VirusTotal, showing that the method proposed can generate adversarial samples in binary format. The contributions of this paper are highlighted as follows.

- (1) This paper proposes a new method of generating adversarial samples by the use of the evolutionary algorithm, which can automatically generate valid adversarial samples
- (2) This paper uses a well-designed feature library as rewriting material in the evolutionary process, which helps generate modified samples with fewer attempts
- (3) This paper applies the adversarial samples generated to attack DL-based malware detection engines on VirusTotal and obtains better experimental results than other attack methods

The rest of this study is organized as follows. Section 2 is a concise introduction of malware detection and adversarial attack methods. Section 3 proposes our attack framework in detail. Section 4 describes the experimental settings and main results and gives a deep analysis. Section 5 concludes this paper as well as the research directions.

2. Related Work

2.1. Machine Learning-Based Malware Detection Methods. In malware detection, machine learning (ML) is a popular approach. Moreover, in recent years, many ML-based malware detection methods have been put forward [21–24]. These methods are mainly categorized in static analysis [25, 26] and dynamic behavior analysis [27, 28]. Static analysis learns the statistical features of malware (e.g., API calls, OpCode), whereas dynamic behavior analysis detects abnormal (possibly malicious) behavior by observing deviations from the baseline of the system. Recently, malware detection efforts prefer to use raw software binaries as the input of DL models [29–31].

NVIDIA’s research group [32] proposed the MalConv network, which took the raw byte sequences of PE files as input directly, achieved 98.88% detection accuracy. Compared with the detection model that extracts only some features of PE files as input, MalConv links other discrete features. Therefore, it can detect samples with arbitrary size and avoid missing important features. This paper assesses the effectiveness and performance of our framework using the MalConv detection system.

2.2. Adversarial Attack against Malware Detection Model. DL-based malware detection approaches are susceptible to adversarial attacks [33–37]. Adversarial modifications by manipulating only a small fraction of raw binary data may lead to misclassification. Moreover, the raw binary contents of data are not changed in a nutshell; otherwise, its originally momentous functionality might lose.

Prior work has proposed various ways of adversarial attack against ML-based malware detection models. Through appending bytes at the end of a binary file while preserving its intrusive functionality, Kolosnjaji et al. [20] designed a gradient-based attack model. However, it is based on white-box attacks and cannot be applied to real scenarios. Kreuk et al. [38] proposed a modification method that injected a minor byte sequence into the originally binary file. It is also based on white-box attacks and is not efficient in real scenarios. Anderson et al. [17] designed an effective model which is based on a deep reinforcement learning method to attack static PE antimalware engines. In their work, the reward function and the environment of reinforcement learning were artificially defined. Later, Fang et al. [39] improved Anderson et al.’s work by autonomously generating the reward function according to the expert strategy. Numerous experiments showed that Fang et al.’s method [39] is more nimble and efficacious than Anderson et al.’s method [17]. Yuan et al. [40] proposed an adversarial sample generation model named GAPGAN. GAPGAN initially maps the discrete malware binaries into a contiguous space;

the output is input to the generator of GAPGAN to generate adversarial payloads. Finally, the generated payloads are appended to the originally binary file to create an adversarial one. Because the valid part of the binary file was not changed, the original functionality of the binary file is preserved. GAPGAN can perform an efficient black-box attack. However, the modification action in GAPGAN involves only a simple action. The GAPGAN cannot perform complicated modifications similar to real malware writers. Song et al. [41] presented a framework for creating adversarial malware and evaluated the evasion capabilities in realistic scenarios. The authors firstly revealed the root causes that adversarial samples evade the malware detection method.

This study puts forward a novel binary manipulation-based attack framework, which generates adversarial samples with an evolutionary learning algorithm. Our method can adaptively simplify the actions of modifying binary samples and use an evolutionary algorithm to make adversarial samples more targeted. The generated adversarial samples by statically rewriting the PE file keep their dynamic behavior consistent and can evade the DL-based malware detection models. Experimental results verify the effectiveness of our method, which can efficiently and quickly generate adversarial samples without human intervention.

3. Adversarial Sample Generation Based on Evolutionary Algorithm

3.1. Problem Description. Our ultimate objective is to generate an antagonistic sample, which manipulates the classifier to classify malicious software as benign by mistake while still retaining the malicious function of the original sample.

Let us consider a classifier f that maps any binary file S into a unique category label. Label = 0 denotes that S is malicious, and label = 1 denotes S is benign. Let $A = \{a_1, a_2, \dots, a_n\}$ be an action set that is used to modify the malware samples. S denotes an original sample, whereas S_{mod} denotes a modified sample. The functionality detecting function v is used to check whether S_{mod} retains the same malicious functionality with S . When the output of the functionality detection function is 1, we consider that S_{mod} retains the malicious functionality of S and then save S_{mod} as the adversarial sample S_{adv} . The specific formulae can be detailed in Section 3.3.3.

In brief, for the malware sample S , our goal is to generate an adversarial sample S_{adv} which makes $f(S_{\text{adv}}) = 1$, and if $v(S, S_{\text{mod}}) = 1$, $S_{\text{adv}} = S_{\text{mod}}$.

3.2. Rewriting Actions. PE file is a generic term for executable files in Windows operating system. A PE file consists of a header, section table, and section data. The MS-DOS header consists of three parts: a DOS header, the true PE header, and an optional header, and it includes some basic messages about the executable file. Section table describes the characteristics of each file section. The section table consists of a series of IMAGE_SECTION_HEADER structures arranged in a sequence. The structures and sections are arranged in a fixed order. Section data consists of 4 main parts: .text, .data, .rdata, and .idata, and the data part includes the prac-

tical contents relating to every section. The PE file format is shown in Figure 1.

For a black-box model, we have no idea of the exact features of the classifier involved. However, by observing the chosen features in some open-sourced classifiers, we can make a wild guess at some of the common features in malware detection models. An adversarial sample is generated by modifying one or several features. The chosen actions of modifying the features should be easy to execute. Moreover, after the features are modified, the executability and functionality of malware should not be corrupted. In this paper, all actions applied to the PE file are shown as follows.

- (1) Appending some bytes to the PE file
- (2) Inserting an unused function to the import address table
- (3) Appending some bytes to the untapped space in a section
- (4) Adding a new section
- (5) Changing a section name
- (6) Packing the file
- (7) Unpacking the file

The malicious binary file is modified through the following steps. Firstly, the original PE file is read, then the content in the specified location is added or deleted, and finally, the relative virtual address of the PE file is modified.

3.3. The Proposed Framework. The workflow of the framework includes three parts: the generation of the feature library, the generation of the modified samples, and the generation of the adversarial samples. Firstly, the feature library is generated using MalGAN. In the processing of generating the adversarial samples, the modified features are randomly selected from the feature library according to the rewriting actions. Then, modified samples by evolutionary algorithms are generated. Finally, the generated modified samples are tested whether the malicious functionality of the original samples remains or not. If a modified sample has the same malicious functionality as the original one, we save it as an adversarial sample. Figure 2 gives an overview of our framework. The details of the three parts are given in the following.

3.3.1. Generation of Feature Library. To efficiently generate adversarial samples, we collect and generate the rewritten feature library using MalGAN. MalGAN, proposed by Hu and Tan [16], is used to generate adversarial samples for attacks based on GAN. The MalGAN architecture primarily consists of three components: a generator, a discriminator, and a black-box detector. By only adding a random number to API calls, MalGAN can transform a malicious feature vector into its opposed version.

Our work is built on this work. Firstly, we construct the sample library consisting of malicious and benign samples and extract their binary features, such as the import

FIGURE 1: PE file format.

FIGURE 2: Overview of the framework.

functions and section names. Then, they are stored in a feature mapping dictionary for convenient retrieval and future operations. Next, we use the feature mapping dictionary to generate separate feature mapping for each malicious and benign sample and send them as the input to the MalGAN. After running for a few epochs, the MalGAN can generate adversarial feature mappings. Finally, according to the adversarial feature dictionary, the adversarial feature mappings are mapped into the feature library.

Once the feature library is generated, the modified features needed are randomly selected from the feature library according to the rewriting operation when generating modified samples.

3.3.2. Generation of Modified Samples Based on Evolutionary Algorithm. Evolutionary algorithms simulate the evolution of species in nature, such as selection, crossover, and mutation, which are often used to solve some optimization problems by choosing the brightest individual from the whole population. Different from traditional optimization algo-

rithms such as calculus-based methods and exhaustive enumeration methods, evolutionary learning is a global optimization algorithm, which is highly robust and widely applicable.

This study uses an evolutionary algorithm for sample rewriting to generate modifications. The evolutionary algorithm can adaptively simplify the actions of modifying samples and make the adversarial sample more targeted. It can efficiently generate modified samples without human intervention. Compared with other existing methods, the evolutionary algorithm starts from the string set, which improves the speed of the algorithm and is easy to parallel computing. There is no backpropagation of weights and biases in deep learning and optimization of the loss function, which decreases the probability of obtaining a local optimum.

In the process of evolution, malware samples are considered individuals. Atomic manipulation of rewriting samples is a gene with a genetic message, and the predictive effect of the detecting model on modified samples is fitness. The generation process of the modified sample is shown in Figure 3.

FIGURE 3: The generation process of a modified sample.

The detailed evolutionary process is as follows.

- (1) Step 1. Population initialization. n segments of the genome are generated randomly
- (2) Step 2. Binary modification. Firstly, n segment genomes from binary sequences are mapped to candidate action sequences. Then, the malware samples are rewritten by candidate action sequences to generate modified samples
- (3) Step 3. Fitness calculation. The modified samples are fed into the MalConv network, and the output of the MalConv is used as the fitness of the individual. A smaller output value of MalConv indicates a higher fitness. The higher the fitness, the higher the probability that a gene sequence will be selected for retention
- (4) Step 4. Selecting the best offspring according to the fitness as the parent of the next generation
- (5) Step 5. Performing genetic manipulation on the selected parents. New offspring through crossover and mutation are reproduced
- (6) Step 6. The assessment of end condition. When the action sequence has reached a minimum value or the maximum number of iterations is reached, the evolution is ended. If the end condition is satisfied, the modified samples are output. Otherwise, skip to Step 2

The detailed process is described in Algorithm 1.

3.3.3. Generation of the Adversarial Samples. It should also be noted that the functionality of a malware sample may be corrupted during the modification process. In other words, its attacking characteristics may be damaged. We consider an adversarial sample without malicious functionality to be invalid. To detect whether the malicious function-

ality of the modified sample is retained, we use the sandbox to collect behaviors of the modified samples and original samples. If the behavior of the modified sample is the same as that of the original one, we think that it retains the malicious function of the original sample, and it is saved as an adversarial sample.

Suppose a behavior of the original sample S is indicated as the set B_S , and behavior of the modified sample S_{mod} is indicated as set B_{mod} . We denote the total number of similar behaviors in B_S and B_{mod} as $\text{num}(B_S, B_{\text{mod}})$ and the size of B_S as $\text{num}(B_S)$. The behavior similarity between B_S and B_{mod} is defined as sm :

$$\text{sm} = \frac{\text{num}(B_S, B_{\text{mod}})}{\text{num}(B_S)}, \quad (1)$$

$$v(S, S_{\text{mod}}) = \begin{cases} 1, & \text{sm} \geq 0.7, \\ 0, & \text{elsewhere.} \end{cases} \quad (2)$$

Because the modification operation is a direct manipulation on the original sample, it can inevitably alter the behaviors of the original samples. Therefore, we assume that if $\text{sm} \geq 0.7$, the samples S and S_{mod} have the same behaviors; that is, $v(S, S_{\text{mod}}) = 1$. It also means that the modified sample retains its original malicious functionality. In the end, we save the modified sample S_{mod} retained originally the malicious functionality as the adversarial sample S_{adv} .

4. Experimental Results and Analysis

This section firstly gives the setting of our experiments, including the datasets, evaluation metrics, and the target malware detection model. Then, we analyze the experiment results.

4.1. Experimental Settings and Evaluation Metrics. In the experiment, we construct a dataset with 1000 malware samples from VirusTotal. Moreover, we also produce some adversarial examples for PE binaries to evaluate the effectiveness of our proposed method.

To assess the effectiveness of adversarial samples, we measure some evaluation metrics in Table 1. N_s denotes the number of modified files that have structural integrity or executability. N denotes the total amount of samples. N_r denotes the number of modified samples that retain the originally malicious functionality. N_e denotes the number of adversarial ones that can evade malware detection engines. The computer specification used for the experiments is as follows: CPU: Intel Core i5-6500, 3.20 GHz, 4 cores, 8 threads; memory: 2 GiB; and operating system: Ubuntu 16.04.

4.2. Experimental Results of Adversarial Attack. This section demonstrates some performances of our approach under antiattack scenes and compares the results with some methods available.

In our experiment, the attacked model is the MalConv proposed by Raff et al. [32]. We train attacked MalConv network using a dataset with 6230 malicious samples from


```

Input: malware samples  $S$ , population scale, number of generations  $G$ .
Output: modified samples  $S_{\text{mod}}$ 
BEGIN
for  $s$  in  $S$  do
    Initialize the population;
    while current generation  $\leq G$  or action sequence is not minimum do
        Map binary sequences to action sequences;
        Modify malware sample based on the action sequences;
        Calculate fitness;
        Select the best offspring;
        Perform crossover;
        Perform mutation;
        Increase current generation;
    end while
    Append the optimal result  $S$  to  $S_{\text{mod}}$ ;
end for
Return  $S_{\text{mod}}$ ;
END

```

ALGORITHM 1: Malware sample evolution.

TABLE 1: Evaluation metrics.

Evaluation metrics	Formula
The success rate of modified sample operation (R_s)	$R_s = N_s/N$
The malicious function retention rate of modified samples (R_r)	$R_r = N_r/N_s$
Evasion rate (R_e)	$R_e = N_e/N_r$

TABLE 2: The performance comparison of adversarial samples generated by different methods.

Attack methods	Evaluation metrics		
	R_s	R_r	Average time
Our method	100%	97.5%	24.5 s
DQEAF	76%	94.3%	60 s
Aut.	68%	95.6%	42.5 s

TABLE 3: The evasion rate of different attack methods against different detection engines.

Detector	Adversarial attack methods		
	DQEAF	Aut.	Our method
ClamAV	19.2%	17.3%	18.5%
Cylance	39.5%	42.5%	47.8%
Endgame	21.3%	22.6%	23.5%
Trampoline	20.6%	18.8%	19.2%

VirusTotal and 5660 benign samples from web crawling and achieve 98.4% detection accuracy.

Furthermore, to explore the validity of the presented binary-based attack method against a deep learning-based detection system, we compare our approach with other byte-level attack methods, including the DQEAF method [39], which is based on reinforcement learning and Aut. method [41], which is based on code randomization and

FIGURE 4: The evasion rate with the generation increase.

TABLE 4: Performance evaluation on MalConv with and without adversarial training.

Defense	Test dataset	Accuracy
No defense	×	98.4%
	√	80.2%
Adversarial training	√	90.5%

× indicates no adversarial samples and √ denotes having adversarial ones in the test dataset.

binary manipulation. The results are shown in Tables 2 and 3, respectively.

Table 2 shows the performance comparison of adversarial samples generated by different methods. From the three evaluation metrics of generated samples, our approach outperforms other similar methods. The reason is that the

selected action of modifying malicious samples is built on ensuring the execution of PE files. The specific modifying actions, such as inserting, changing, and adding actions, are taken from the generated feature library. The feature library has also been carefully designed to ensure the authenticity of all actions. Therefore, the modification does not involve invalid actions which damage the primary structure of malware or lose its original functionality. Meanwhile, our action set does not cover the irreversible actions, such as removing signatures, which makes our action set is more effective. Our approach can do a heuristic random search which simplifies the modification actions and does not require performing action sequence minimization and marking action weights and success content as the other two methods, which significantly raises the efficiency of our method.

To further test the effectiveness, we evaluate it using four representative malware detection engines on VirusTotal, including ClamAV, Cylance, Endgame, and Trapmine. Table 3 demonstrates different evasion rates of adversarial samples produced by some methods against different detection engines. From Table 3, we can see that the evasion rate of our method has a better performance compared with the other two in most cases. Meanwhile, to test the efficiency, we record the evasion rate with the generation increase, shown in Figure 4. From Figure 4, we can find that our method reaches a relatively stable evasion rate after 15 generations, which shows that our method is very efficient.

4.3. Defense against Adversarial Sample Attack. To defend against adversarial attacks, more and more defense countermeasures have been proposed. Among them, adversarial training is one of the most popular ways [42], in which adversarial samples are added to the training set; thus, DL models can adjust the decision strategies. Compared with other adversarial defense methods, adversarial training does not require modifying the detection model and is easy to implement.

In this paper, we use adversarial samples generated by evolutionary algorithms to test on MalConv network. Table 4 shows the performance evaluation on MalConv with and without adversarial training. The experimental results show that the detection accuracy increased from 80.2% to 90.5% after the adversarial training. It also illustrates that adversarial training can effectively improve the model robustness to adversarial attacks.

5. Conclusion

To make DL-based IoT malware detection models more robust and effective, we propose a framework for generating adversarial samples and their defense. Our framework firstly adopts an evolutionary algorithm to generate modified samples, and then, the modified samples that retain the originally malicious functionality are saved as adversarial samples. This method does not need to obtain any information of the special detection models containing extracted features, internal parameters, etc. Moreover, our approach is entirely automated without human intervention. The exper-

imental results demonstrate that our method can ensure the diversity of generated samples and greatly enhance the efficiency of adversarial sample generation. This paper also demonstrates that adversarial training is one of the effective methods to combat adversarial sample attacks.

The action space has a great influence on the diversity and versatility of evolutionary optimization algorithms. Defining more effective modification actions to expand the search space of evolutionary algorithms is our urgent task. Our future work also includes accelerating the convergence speed and improving the stability of the evolutionary algorithm. Moreover, we will also explore more methods of generating adversarial samples to defend against adversarial attacks on IoT.

Data Availability

The dataset can be obtained from the website: <https://www.virustotal.com> (accessed on 22 April 2021).

Conflicts of Interest

The authors declare no conflict of financial or associative interest in connection with the manuscript submitted.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61572170), the Natural Science Foundation of Hebei Province of China (Grant Nos. F2019205163 and F2021205004), the Science and Technology Foundation Project of Hebei Normal University (Grant No. L2021K06), the Science Foundation of Returned Overseas of Hebei Province of China (Grant No. C2020342), the Science Foundation of Department of Human Resources and Social Security of Hebei Province (Grant Nos. 201901028 and ZD2021062), and the Natural Science Foundation of Hebei Normal University (Grant No. L072018Z10).

References

- [1] J. Qiu, Z. Tian, C. du, Q. Zuo, S. Su, and B. Fang, "A survey on access control in the age of Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.
- [2] Y. Xu, C. Zhang, Q. Zeng, G. Wang, J. Ren, and Y. Zhang, "Blockchain-enabled accountability mechanism against information leakage in vertical industry services," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1202–1213, 2021.
- [3] X. Yan, Y. Xu, X. Xing, B. Cui, Z. Guo, and T. Guo, "Trustworthy network anomaly detection based on an adaptive learning rate and momentum in IIoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6182–6192, 2020.
- [4] A. Sharma, P. Malacaria, and M. H. R. Khouzani, "Malware detection using 1-dimensional convolutional neural networks," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroSec&PW)*, pp. 247–256, Stockholm, Sweden, 2019.

- [5] X. Pei, L. Yu, and S. Tian, "AMalNet: a deep learning framework based on graph convolutional networks for malware detection," *Computers & Security*, vol. 93, p. 101792, 2020.
- [6] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, pp. 344–367, 2021.
- [7] S. Yoo, S. Kim, S. Kim, and B. B. Kang, "AI-HydRa: advanced hybrid approach using random forest and deep learning for malware classification," *Information Sciences*, vol. 546, no. 9, pp. 420–435, 2021.
- [8] Y. Xu, J. Ren, Y. Zhang, C. Zhang, B. Shen, and Y. Zhang, "Blockchain empowered arbitrable data auditing scheme for network storage as a service," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 289–300, 2019.
- [9] Y. Xu, Q. Zeng, G. Wang, C. Zhang, J. Ren, and Y. Zhang, "An efficient privacy-enhanced attribute-based access control mechanism," *Concurrency & Computation Practice & Experience*, vol. 32, no. 5, 2020.
- [10] 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, USA, 2015.
- [11] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, Paris, France, 2018.
- [12] X. Liu, J. Zhang, Y. Lin, and H. Li, "ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples," in *IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, Phoenix Arizona, USA, 2019.
- [13] H. Rathore, S. K. Sahay, P. Nikam, and M. Sewak, "Robust Android malware detection system against adversarial attacks using q-learning," *Information Systems Frontiers*, vol. 22, no. 8, pp. 1–20, 2020.
- [14] C. Zhang, Y. Xu, Y. Hu, J. Wu, J. Ren, and Y. Zhang, "A blockchain-based multi-cloud storage data auditing scheme to locate faults," *IEEE Transactions on Cloud Computing*, 2021.
- [15] D. Li, Q. Li, Y. Ye, and S. Xu, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 736–750, 2021.
- [16] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, <http://arxiv.org/abs/1702.05983>.
- [17] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static PE machine learning malware models via reinforcement learning," 2018, <http://arxiv.org/abs/1801.08917>.
- [18] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," 2016, <http://arxiv.org/abs/1606.04435>.
- [19] O. Suci, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14, San Francisco, CA, USA, 2019.
- [20] B. Kolosnjaji, A. Demontis, B. Biggio et al., "Adversarial malware binaries: evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 533–537, Rome, Italy, 2018.
- [21] M. Shafiq, Z. Tian, A. K. Bashir, X. du, and M. Guizani, "CorAUC: a malicious bot-IoT traffic detection method in IoT network using machine-learning techniques," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3242–3254, 2021.
- [22] S. Su, Z. Tian, S. Liang, S. Li, S. du, and N. Guizani, "A reputation management scheme for efficient malicious vehicle identification over 5G networks," *IEEE Wireless Communications*, vol. 27, no. 3, pp. 46–52, 2020.
- [23] C. Luo, Z. Tan, G. Min, J. Gan, W. Shi, and Z. Tian, "A novel web attack detection system for Internet of Things via ensemble classification," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5810–5818, 2021.
- [24] M. Shafiq, Z. Tian, A. Bashir, A. Jolfaei, and X. Yu, "Data mining and machine learning methods for sustainable smart cities traffic classification: a survey," *Sustainable Cities and Society*, vol. 60, 2020.
- [25] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, Article ID 7247095, 16 pages, 2018.
- [26] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ML-based visualization technique," *Computers & Security*, vol. 89, 2020.
- [27] W. Huang and J. W. Stokes, "MtNet: a multi-task neural network for dynamic malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 399–418, Springer, 2016.
- [28] 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*, pp. 137–149, Springer, 2016.
- [29] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Generation Computer Systems*, vol. 105, no. 4, pp. 230–247, 2020.
- [30] H. Guo, S. Huang, C. Huang et al., "A lightweight cross-version binary code similarity detection based on similarity and correlation coefficient features," *IEEE Access*, vol. 8, pp. 120501–120512, 2020.
- [31] N. A. Azeez, O. E. Odufuwa, S. Misra, J. Oluranti, and R. Damaševičius, "Windows PE malware detection using ensemble learning," *Informatics*, vol. 8, no. 1, pp. 10–20, 2021.
- [32] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole EXE," 2017, <http://arxiv.org/abs/1710.09435>.
- [33] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Black-box adversarial sample generation based on differential evolution," *Journal of Systems and Software*, vol. 170, no. 8, Article 110767, pp. 1–11, 2020.
- [34] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Computing and Applications*, vol. 32, no. 18, pp. 14781–14800, 2020.
- [35] M. Li, Y. Sun, H. Lu, S. Maharjan, and Z. Tian, "Deep reinforcement learning for partially observable data poisoning

- attack in crowdsensing systems,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6266–6278, 2020.
- [36] R. Podschwadt and H. Takabi, “On effectiveness of adversarial examples and defenses for malware classification,” in *International Conference on Security and Privacy in Communication Systems*, pp. 380–393, Springer, 2019.
- [37] D. Maiorca, B. Biggio, and G. Giacinto, “Towards adversarial malware detection,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.
- [38] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, “Deceiving end-to-end deep learning malware detectors using adversarial examples,” 2018, <http://arxiv.org/abs/1802.04528>.
- [39] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, and H. Huang, “Evading anti-malware engines with deep reinforcement learning,” *IEEE Access*, vol. 7, pp. 48867–48879, 2019.
- [40] J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, “Black-box adversarial attacks against deep learning based malware binaries detection with GAN,” in *the 24th European Conference on Artificial Intelligence (ECAI 2020)*, pp. 2536–2542, Santiago de Compostela, Spain, 2020.
- [41] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, “Automatic generation of adversarial examples for interpreting malware classifiers,” 2020, <http://arxiv.org/abs/2003.03100>.
- [42] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2015, <http://arxiv.org/abs/1412.6572>.