IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 31, 2020, accepted July 7, 2020, date of publication July 10, 2020, date of current version July 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3008433

A Brute-Force Black-Box Method to Attack Machine Learning-Based Systems in Cybersecurity

SICONG ZHANG[1,2], XIAOYAO XIE[2], (Member, IEEE), AND YANG XU[2]

[1]School of Computer Science and Technology, Guizhou University, Guiyang 550025, China
[2]Key Laboratory of Information and Computing Science Guizhou Province, Guizhou Normal University, Guiyang 550001, China

Corresponding author: Sicong Zhang (351625648@qq.com)

This work was supported in part by the National Natural Science Foundation of China under Grant U1831131, in part by the Central Government Guides Local Science and Technology Development Special Funds under Grant [2018]4008, in part by the Technology Cooperation Key Project of Guizhou Province, China Grant [2015]7763, and in part by the Science and Technology Planned Project of Guizhou Province, China under Grant [2020]2Y013.

ABSTRACT Machine learning algorithms are widely utilized in cybersecurity. However, recent studies show that machine learning algorithms are vulnerable to adversarial examples. This poses new threats to the security-critical applications in cybersecurity. Currently, there is still a short of study on adversarial examples in the domain of cybersecurity. In this paper, we propose a new method known as the brute-force attack method to better evaluate the robustness of the machine learning classifiers in cybersecurity against adversarial examples. The proposed method, which works in a black-box way and covers some shortages of the existing adversarial attack methods based on generative adversarial networks, is simple to implement and only needs the output of the target classifiers to generate adversarial examples. To have a comprehensive evaluation of the attack performance of the proposed method, we use our method to generate adversarial examples against the common machine learning based security systems in cybersecurity including host intrusion detection systems, Android malware detection systems, and network intrusion detection systems. We compare the attack performance of the proposed method against these security systems with that of state-of-the-art adversarial attack methods based on generative adversarial networks. The preliminary experimental results show that the proposed method, which is more efficient in computation and outperforms the state-of-the-art attack methods based on generative adversarial networks, can be used to evaluate the robustness of various machine learning based systems in cybersecurity against adversarial examples.

INDEX TERMS Adversarial examples, machine learning, deep learning, intrusion detection, malware detection, neural networks, black-box method.

I. INTRODUCTION

Most scenarios in cybersecurity, such as malware detection [1] and intrusion detection [2], can be viewed as classification problems. Machine learning is effective in classification issues and shows excellent performance in cybersecurity, so it is widely applied in this domain [3]. However, with the emergence of adversarial examples [4], the machine learning based systems in this security-critical field face new challenges. Adversarial examples (AEs), which are generated by adding intentionally crafted noises to the original inputs, can make the target classifiers misclassify. In the context of cybersecurity, this usually means adversaries disguise malicious behavior as normal to evade the detection of the machine learning based systems.

Currently, the research on AEs mainly focuses on computer vision [4]–[7]. Although there has been some pioneering work [8]–[11] about AEs in cybersecurity, there is still a lack of relevant research in this domain. On the other hand, the previous studies [7]–[11] concentrate on generating the AEs against deep neural networks. The research on AEs against traditional machine learning algorithms in cybersecurity is less concerned. Therefore, to better evaluate the robustness of the machine learning based security systems in cybersecurity, it is necessary to further research the generation of AEs against machine learning classifiers in this domain.

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Xiao.

Generative adversarial networks (GANs) [12] are now the hot topic in deep learning. GAN has shown excellent performance in generating images, sounds, and texts [13]. GANs are a new generative framework that is composed of the discriminator and the generator. Normally, the discriminator and the generator are both neural networks. The discriminator learns to distinguish whether the inputs are fake or real. The generator learns to produce high-quality fake samples which can mislead the discriminator. Through the competition between the discriminator and the generator, GAN can be trained to produce real-like samples. Because of its excellent generating performance, it becomes one of the most prevailing and effective methods to generate AEs [9], [14], [15] in cybersecurity. Although the adversarial attack methods based on GANs can make the target classifiers misclassify in a high success rate, the training of GANs is currently unstable [9], [15] and the attack performance of GAN-based methods is influenced by training data [14], which are not always available for adversaries. Besides, GANs are normally composed of two deep neural networks. Neural networks are generally regarded as black-box models, which means the process of generating AEs in GANs is uncontrolled by the adversaries. We can not intervene in the generating process to decide the features to be perturbed when generating AEs with GANs. This will hinder the application of adversarial attack methods based on GANs under certain circumstances in cybersecurity because sometimes we can only perturb some specific features of input vectors to ensure that the functionality of the inputs does not change. This will be discussed detailedly in Section III.

In this paper, to avoid the tedious training of the GAN-based adversarial attack methods and generate adversarial examples more efficiently, we propose a new and simple black-box attack method known as the brute-force attack method (BFAM) to better evaluate the robustness of the machine learning based systems in cybersecurity against AEs. Our method is simple to implement compared with the GAN-based methods. Our method modifies the features of input vectors for machine learning based systems in a controlled way. Our method does not need the internal information of the target classifiers and is a gradient-free method, which means the computation of the gradient is unnecessary. The outputs of the target classifiers are the only needed knowledge for generating AEs. Specifically, our method needs the confidence scores of the target classifiers. Our method can be used to produce AEs against different machine learning based systems in cybersecurity. To validate this, we design three experiments involving different scenarios of cybersecurity, i.e., host intrusion detection [16], network intrusion detection [2], and Android malware detection [17], where the machine learning algorithms are widely used to improve the detection performance of the target systems. Our method operates in a black-box way, which is more common in the real world because the adversary can only obtain the output of the target classifier in most cases. The internal knowledge of the target classifiers is usually unknown to the adversary. To verify the attack performance of our method against various machine learning algorithms, we adopt the machine learning algorithms which are widely applied in cybersecurity to build up the target systems, i.e., logistic regression (LR) [15], [17], decision tree (DT) [2], [3], [16], [17], multilayer perceptron (MLP) [2], [17], naive Bayes (NB) [2], [3], [17] and random forest (RF) [2], [17]. In general, we make the following contributions:

- We propose a new method known as the brute-force attack method to generate AEs against machine learning based systems in cybersecurity. Our method is simple to implement and avoids the tedious training of GAN-based attack methods. Besides, our method is a gradient-free method and manipulates the features of input vectors in a determinate way, which makes our method more suitable for the adversarial attacks in cybersecurity. Our method generates AEs based on the confidence scores of the target classifiers heuristically.
- Android malware detection, host intrusion detection, and network intrusion detection are common scenarios of cybersecurity, where machine learning techniques are widely used. We compare the attack performance of BFAM against different target classifiers in these scenarios with that of the state-of-the-art GAN-based attack methods to have a comprehensive evaluation of our method.
- Our method is a black-box attack method for which the architectures and parameters of the target classifiers are unnecessary for attacks. The confidence scores of the target models are the only required knowledge to produce AEs. The most widely applied machine learning classifiers for different scenarios of cybersecurity are chosen as our target classifiers. Through this, we hope to have a full-scale evaluation of the robustness of different machine learning classifiers against AEs.

II. RELATED WORK

In recent years, machine learning has shown promising results in the field of cybersecurity. Machine learning algorithms are diffusely adopted to solve various tasks in this domain including malware detection [1], [17], intrusion detection [2], [3], [16], spam filtering [18], *et al.* Fatima *et al.* [19] utilize the genetic algorithm to select features and use the selected features to train machine learning based Android malware classifiers. Their results show that the feature dimension can be reduced to half of the original feature set with the help of the genetic algorithm and high accuracy can be achieved. Yao *et al.* [20] adopt machine learning techniques to propose a new intrusion detection framework to overcome the imbalance of different kinds of data in network traffic and the nonidentical distribution between the training set and the test set. Ren *et al.* [21] adopt isolation forest, genetic algorithm, and RF to design a new intrusion detection system which mainly consists of data sampling and feature selection. Vijayakumar and Ganapathy [22] explore

the role of machine learning to reduce the false alarm rate of wireless intrusion detection systems. They propose a new filtration technique to reduce false alarms. Sahın *et al.* [23] evaluate the spam classification performance of different machine learning methods combined with the bag of words technique.

Because machine learning is widely applied in cybersecurity, it is necessary to pay more attention to AEs, which pose new threats to the machine learning based systems in this domain and restrict the further application of machine learning algorithms in this security-critical field. Szegedy *et al.* [4] are the first to reveal the vulnerability of deep neural networks to AEs. Small but intentionally crafted noises added to original inputs can make the target classifiers misclassify. They adopt the box-constrained limited memory approximation of the Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to generate AEs against the classifiers in computer vision successfully. Papernot *et al.* [5] make use of the Jacobian matrix of the target classifiers to determine the input features to be perturbed. Their method is called the Jacobian-based saliency map attack (JSMA). Moosavi-Dezfooli *et al.* [6] propose DeepFool to produce AEs by finding the closest distances between original inputs and the decision boundary. Carlini and Wagner [7] put forward three new gradient-based attack methods by introducing new objective functions. The Carlini and Wagner (CW) attacks are claimed to be more effective than many previously proposed attack methods. The above methods are all white-box attacks.

With machine learning techniques widely applied in cybersecurity, more and more attention is paid to the study of AEs in cybersecurity. Grosse *et al.* [8] are the first to research the generation of AEs in cybersecurity. They expand the JSMA to generate AEs against the deep learning based Android malware detection systems (AMDSs). Yang *et al.* [9] investigate the attack performance of common black-box adversarial attacks against deep learning based network intrusion detection systems (NIDSs). They adopt three kinds of black-box attack methods including zeroth-order optimization attacks, training a substitute model, and GAN-based attack methods to evaluate the robustness of deep neural networks for NIDS. Wang [10] assesses the attack performance of state-of-the-art adversarial attack methods against deep learning based intrusion detection systems including JSMA, DeepFool, and CW. Liu *et al.* [11] explore the generation of AEs based on the genetic algorithm in the domain of the internet of things (IoT).

Because of the excellent generating performance of GANs, GANs are widely adopted to generate AEs against various security systems in cybersecurity. Hu and Tan [14] propose a new framework called MalGAN to generate adversarial malware to mislead machine learning based malware detection systems. Lin *et al.* [15] propose IDSGAN to produce AEs against machine learning based network intrusion detection systems. The adversarial attack methods based on GANs are currently one of the most widely used [9], [14], [15] and effective [9] black-box attacks in cybersecurity.

III. BACKGROUND

In this section, we briefly introduce the background knowledge about this paper, which will help understand the subsequent sections. It mainly involves AEs, GANs, adversarial attacks based on GANs, and the datasets used in our experiments.

A. ADVERSARIAL EXAMPLES

AEs are derived from adding small but intentionally crafted perturbations to original inputs. The objective of AEs is normally to make the target classifiers misclassify. In computer vision, the perturbations added to inputs are needed to be imperceptible to human eyes. In cybersecurity, this restriction is usually replaced by ensuring that the functionality of the adversarial examples is unchanged [8], [15], [24]. Assuming that the purpose of malware is to steal the password of the target system, it is meaningless that the adversaries generate the adversarial malware using the adversarial attack methods, which can evade the detection of the target security system but lose the ability to steal the password. Therefore, it is necessary to guarantee the validity of AEs.

The adversarial attack methods of generating AEs can be categorized as white-box and black-box based on the knowledge of the target classifiers possessed by the adversaries. The white-box attacks normally assume that the adversaries possess the complete knowledge of the target classifiers including internal parameters, architectures, training data, *et al.* [24]. The black-box attacks assume that the adversaries own limited knowledge of the target classifiers, which does not include the model parameters [24].

The adversarial attack methods can also be categorized as targeted and non-targeted based on the adversarial goal. Let $C(\boldsymbol{x})$ be the label predicted by the classifier for $\boldsymbol{x}$, $\boldsymbol{x}$ be the original input, $\boldsymbol{x}^*$ be the corresponding adversarial example, y be the true label of $\boldsymbol{x}$, and y^* be the target label which the adversaries want the target classifier to output. The goal of non-targeted attacks is to find a $\boldsymbol{x}^*$ which makes the classifier output $C(\boldsymbol{x}^*) \neq y$. The goal of targeted attacks is to make the classifier output $C(\boldsymbol{x}^*) = y^*$.

B. GENERATIVE ADVERSARIAL NETWORKS

GANs are a new category of generative models that are originally proposed by Goodfellow *et al.* [12]. GANs show excellent performance on various tasks including image translation, semi-supervised learning, image generation, *et al.* [13]. GANs mainly consist of two important components, i.e., the discriminator and the generator. The generator takes in the latent variable $\boldsymbol{z}$ as inputs and learns how to produce high-quality fake samples to fool the discriminator. On the contrary, the discriminator takes in fake samples generated by the generator and real samples as inputs and learns how to distinguish between real samples and fake samples. Through the competition between the discriminator and the generator, GANs generate high-quality real-like samples [13]. The normal architecture of GANs is shown

in Fig. 1, where z is the latent variable usually sampled from a uniform distribution.

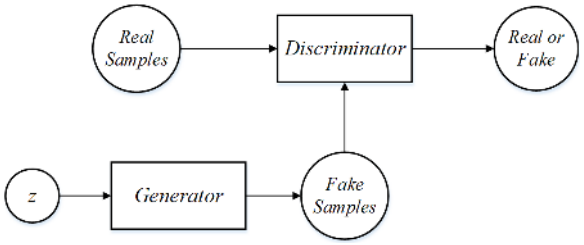

FIGURE 1. The normal architecture of GANs.

The training of GANs can be viewed as a game between the generator and the discriminator. On each training step, the discriminator is first trained to distinguish whether the input is real or fake. Then the generator is trained to produce fake samples that can fool the discriminator. By iteratively carrying out this process, if the discriminator and the generator have the competent capacity, the Nash equilibrium will be achieved [12], [13]. Then the discriminator can not distinguish whether the input is real or fake.

Generally, when the discriminator is trained, the generator needs to be fixed and when the generator is trained, the discriminator needs to be fixed. The training of the discriminator is usually to minimize the loss function in (1). The loss function (1) for the discriminator can be regarded as a standard cross-entropy loss used to train a standard binary classifier. The difference is that the discriminator is trained on two batches of data [12], [13]. To minimize the loss function (1) corresponds to training the discriminator to distinguish between the real samples from the real data distribution, which are labeled as 1, and the fake samples generated by the generator, which are labeled as 0. The training of the generator is usually to minimize the loss function in (2). The loss function (2) for the generator is minimized to make the generator produce the fake samples predicted as real by the discriminator. In these two equations, G denotes the generator and D denotes the discriminator. $\boldsymbol{x}$ denotes the real samples from the real data distribution p_{data} and z denotes the latent variable from the distribution p_z.

$$J^D = -\frac{1}{2}E_{\boldsymbol{x}\sim p_{data}}\log D(\boldsymbol{x}) - \frac{1}{2}E_{z\sim p_z}\log(1 - D(G(z))). \quad (1)$$

$$J^G = -\frac{1}{2}E_{z\sim p_z}\log D(G(z)). \quad (2)$$

C. ADVERSARIAL ATTACKS BASED ON GANs

The adversarial attack methods based on GANs are currently the most concerned black-box attacks in cybersecurity [9], [14], [15]. The advantages of adversarial attacks based on GANs are as follows: 1) The GAN-based adversarial attacks only require the predicted labels of the target classifiers to generate AEs, which makes them applicable for more tasks. The internal information of the target classifiers is unknown to the adversaries most of the time. The white-box attacks do not work in this situation. 2) Although the GAN-based attack methods need minimal knowledge of the target classifiers, AEs generated by them show an excellent result in misleading the target models [9], [14], [15].

However, the training of the GANs is currently still unstable [9], [14], [15]. The attack performance of GAN-based methods partly depends on the training data [14]. The generating process of GAN-based methods is normally uncontrolled because the generator is usually deep neural networks which are regarded as black-box models.

In cybersecurity, the existing adversarial attack methods based on GANs adopt a similar architecture to generate AEs, which is shown in Fig. 2. The generator takes the concatenation of malicious examples and noise as inputs to produce adversarial malicious examples. The black-box detector in Fig. 2 is the target classifier the adversaries want to attack. The black-box detector is just employed to provide the labels of benign examples and adversarial malicious examples for the discriminator. In other words, the discriminator is used to fit the black-box detector and provide the indirect knowledge for the generator to produce the AEs which can deceive the black-box detector.

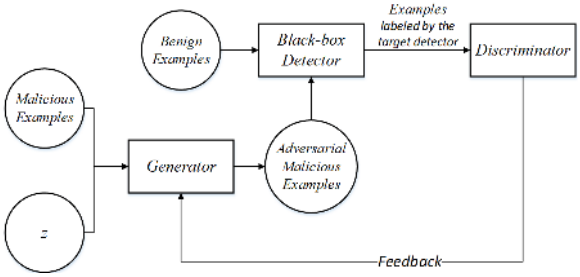

FIGURE 2. The normal architecture of adversarial attacks based on GANs.

The loss functions for the discriminator and the generator are changed to (3) and (4) [14], [15]. In (3) and (4), $\boldsymbol{S}_{benign}$ and $\boldsymbol{S}_{admal}$ denote benign examples and adversarial malicious examples predicted by the black-box detector. $\boldsymbol{S}_{malicious}$ denotes the original malicious examples in the dataset. D denotes the discriminator and G denotes the generator. $\boldsymbol{x}$ is the input for the discriminator and the generator. z is the noise vector from the distribution p_z which is usually a uniform distribution. To train this framework to generate AEs, the loss functions in (3) and (4) need to be minimized.

$$J^D = E_{\boldsymbol{x}\in\boldsymbol{S}_{benign}}D(\boldsymbol{x}) - E_{\boldsymbol{x}\in\boldsymbol{S}_{admal}}D(\boldsymbol{x}). \quad (3)$$

$$J^G = E_{\boldsymbol{x}\in\boldsymbol{S}_{malicious},z\in p_z}D(G(\boldsymbol{x},z)). \quad (4)$$

D. DATASETS

To have a comprehensive evaluation of the attack effect of our method, three datasets which involve three scenarios of cybersecurity are adopted to validate the effectiveness of our method.

The Australian Defence Force Academy Linux Dataset (ADFA-LD) is a professional and widely used host intrusion detection dataset [16], [25] published by Creech and Hu [26]. The system call traces are usually used by the host intrusion detection systems (HIDSs) to detect the attacks against the target systems. ADFA-LD consists of 833 normal training traces, 4372 normal validation traces, and 746 attack traces, which are all collected under the Linux system. Each system call in the traces is represented by an integer. The details of the ADFA-LD dataset are shown in Table 1.

TABLE 1. Details of the ADFA-LD dataset.

Data Type		Traces
Normal Data	Training Data	833
	Validation Data	4372
Attack Data	Adduser	91
	Hydra_FTP	162
	Hydra_SSH	176
	Java_Meterpreter	124
	Meterpreter	75
	Web Shell	118

The NSL-KDD dataset is a benchmark dataset for network intrusion detection, which is widely adopted to evaluate the performance of NIDSs [2], [16], [20]. The previous work [9], [10], [15] mostly adopts the NSL-KDD dataset to verify the attack effect of their methods. Every record in NSL-KDD is composed of 41 features which can be grouped into four feature sets: Intrinsic, Content, Time-based Traffic, and Host-based Traffic. Each record in NSL-KDD is labeled as normal or a specific attack type. There are four main attacks, i.e., Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L). The four kinds of attacks can be further divided into 38 attack classes. The training data contain 22 attack classes and the testing data contain 37 attack classes, in which 16 novel attacks only exist in the test set. The details of NSL-KDD are shown in Table 2.

TABLE 2. Details of the NSL-KDD dataset.

NSL-KDD		Training	Testing
Normal		67343	9711
Attack	DoS	45927	7458
	Probe	11656	2421
	R2L	995	2887
	U2R	52	67
Total		125973	22544

The DREBIN dataset originally published by Arp *et al.* [27] is a widely used Android malware detection dataset [8]. DREBIN includes 129013 Android applications, which consist of 123453 benign applications and 5560 malicious applications. Each application can be transformed into a vector with 54533 features, each of which is represented by a binary value to indicate whether the feature is present in the application [8]. All the features can be divided into 8 feature sets. The detailed statistics of the number of features in each feature set is shown in Table 3.

TABLE 3. Details of the DREBIN dataset.

Feature Set	Name	Number of Features
S_1	Hardware Components	72
S_2	Permissions	3812
S_3	Components	223464
S_4	Intents	6379
S_5	Restricted API Calls	315
S_6	Used Permissions	70
S_7	Suspicious API Calls	733
S_8	Network Addresses	310488

IV. PROPOSED METHOD

The generation of AEs in a targeted way can usually be formalized as solving the optimization problem in (5) [4]–[11], where $\boldsymbol{x}$ is the original input, $C(\boldsymbol{x})$ denotes the label predicted by the classifier for $\boldsymbol{x}$, y^* is the target label and $\boldsymbol{\delta}$ is the minimal adversarial perturbation causing the target classifier to misclassify. $\boldsymbol{\delta}$ is normally not unique. $\|\cdot\|_p$ denotes a norm.

$$\begin{aligned} &\text{minimize } \|\boldsymbol{\delta}\|_p \\ &\text{s.t. } C(\boldsymbol{x}+\boldsymbol{\delta}) = y^*. \end{aligned} \tag{5}$$

The adversarial perturbation $\boldsymbol{\delta}$ needs to be small enough to be imperceptible to human eyes in computer vision. As discussed in Section III.A, this restriction is replaced by guaranteeing the functionality of the malware or intrusion data when AEs are generated in cybersecurity, which implies that the adversarial perturbation $\boldsymbol{\delta}$ added to the original inputs does not need to be small but it can not destroy the original functionality of AEs [8]. Therefore, the distortion of the adversarial examples is normally not concerned in cybersecurity [9], [10], [14], [15], [24].

Generally, because of the nonconvexity and nonlinearity of the target classifiers, there is no closed-form solution to the problem in (5). Therefore, many existing attack methods adopt optimization algorithms such as the gradient-based method to achieve the approximate adversarial perturbation. Inspired by the previous work [4]–[11], [14], [15], [28], [29], [30], the generation of AEs can also be viewed as the problem to search the key features of an input vector which can affect the output of the target classifiers. By modifying these key features appropriately, the AEs can be produced to fool the target classifiers. In the setting of black-box attacks, the generation of AEs is then transformed into searching the key features with limited knowledge. Assuming that the confidence scores of the target classifiers are the only information that adversaries can obtain from the target classifiers, the problem is further transformed into searching the key features under the instruction of the confidence scores of the target classifiers.

The intuitive attempt is to modify the features in the input vector successively and determine which modification helps generate AEs which can mislead the target classifiers. This is like the situation that when a hacker wants to make illegal access to a target system without knowing the login password, he will try every possible combination of the passwords and determine the right login password based on the response of

the target system. Inspired by the attack mode of this common attack in cybersecurity, we propose the brute-force attack method (BFAM) to generate AEs against machine learning based systems in cybersecurity.

Before the detailed introduction to BFAM, some notations need to be elaborated first. Assuming that $F(\boldsymbol{x})$ is the confidence scores outputted by the target classifier. $F_i(\boldsymbol{x})$ is the ith component of $F(\boldsymbol{x})$ and indicates the probability of $\boldsymbol{x}$ belonging to Class i. $0 \leq F_i(\boldsymbol{x}) \leq 1$ and $\sum_i F_i(\boldsymbol{x}) = 1$. Therefore, the label predicted by the target classifier is $C(\boldsymbol{x}) = \text{argmax}_i F_i(\boldsymbol{x})$.

Algorithm 1 Targeted Version of Brute-Force Attack Method

Input: Target classifier F; original input $\boldsymbol{x}$; target label y^*; perturbation strength α; indexes of features allowed to be perturbed ***p_index***.
Output: Adversarial examples ***adx***.

$i \leftarrow 0$;
while $\text{argmax}_j F_j(\boldsymbol{x}) != y^*$ **and** $i < \text{len}(\boldsymbol{p\_index})$ **do**
  $\boldsymbol{f}$ stores the confidence scores before modification: $\boldsymbol{f} \leftarrow F(\boldsymbol{x})$;
  perturb the feature $\boldsymbol{x}[\boldsymbol{p\_index}[i]]$ with the strength α: $\boldsymbol{x}[\boldsymbol{p\_index}[i]] \leftarrow \boldsymbol{x}[\boldsymbol{p\_index}[i]] + \alpha$;
  clip the $\boldsymbol{x}$ into the allowed range;
  recalculate the confidence score $F(\boldsymbol{x})$;
  if $F_{y^*}(\boldsymbol{x}) - \boldsymbol{f}[y^*] > 0$ **then**
    keep the modification;
  else
    cancel the modification, return the feature to its original value;
  end if
  $i \leftarrow i + 1$;
end while
adx $\leftarrow \boldsymbol{x}$;
return ***adx***.

The intact pseudocode of the targeted version of BFAM is shown in Algorithm 1. BFAM has two hyperparameters, i.e., ***p_index*** and α. ***p_index*** stores the indexes of the features of $\boldsymbol{x}$ which are permitted to be modified. As discussed in Section III.A, we need to guarantee the functionality of the adversarial examples. Therefore, we can only modify the nonfunctional features of the input vectors [8], [15]. In each iteration, the confidence scores before the modification are first stored in $\boldsymbol{f}$. Then, we modify a permitted feature in $\boldsymbol{x}$ with a specified perturbation strength α. Then, the modified feature needs to be clipped into its original value range in case of destroying the functionality of the input. Through the clipping, the feature which is less than the minimum value is set to the minimum value and the feature which is greater than the maximum value is set to the maximum value. The modified $\boldsymbol{x}$ is fed to the target classifier to recalculate the confidence scores $F(\boldsymbol{x})$. If the $F_{y^*}(\boldsymbol{x}) - \boldsymbol{f}[y^*]$ is greater than 0, the modification to the current feature is valid. The modification is kept. Otherwise, we cancel the modification to the current feature and restore the feature to its original value. $\boldsymbol{f}[y^*]$ denotes the original confidence score of the target class. Two conditions will terminate the while loop: 1) The AE which can mislead the target classifier has been produced. 2) All the features which are allowed to be perturbed have been tried.

The nontargeted version of BFAM is shown in Algorithm 2. The biggest difference between the targeted BFAM and the nontargeted BFAM is that the targeted BFAM is to increase the confidence score of the target class, but the nontargeted BFAM is to decrease the confidence score of the real class.

Algorithm 2 Non-Targeted Version of Brute-Force Attack Method

Input: Target classifier F; original input $\boldsymbol{x}$; true label y; perturbation strength α; indexes of features allowed to be perturbed ***p_index***.
Output: Adversarial examples ***adx***.

$i \leftarrow 0$;
while $\text{argmax}_j F_j(\boldsymbol{x}) == y$ **and** $i < \text{len}(\boldsymbol{p\_index})$ **do**
  $\boldsymbol{f}$ stores the confidence scores before modification: $\boldsymbol{f} \leftarrow F(\boldsymbol{x})$;
  perturb the feature $\boldsymbol{x}[\boldsymbol{p\_index}[i]]$ with the strength α: $\boldsymbol{x}[\boldsymbol{p\_index}[i]] \leftarrow \boldsymbol{x}[\boldsymbol{p\_index}[i]] + \alpha$;
  clip the $\boldsymbol{x}$ into the allowed range;
  recalculate the confidence score $F(\boldsymbol{x})$;
  if $\boldsymbol{f}[y] - F_y(\boldsymbol{x}) > 0$ **then**
    keep the modification;
  else
    cancel the modification, return the feature to its original value;
  end if
  $i \leftarrow i + 1$;
end while
adx $\leftarrow \boldsymbol{x}$;
return ***adx***.

As shown in Algorithm 1 and Algorithm 2, our method does not utilize neural networks to generate the adversarial examples. The proposed method determines the features to be perturbed based on the confidence scores outputted by the target classifiers and produces the adversarial examples by directly manipulating the features of input vectors. We can regard the generating process of our method as that the confidence scores indicate the direction along which we should modify the inputs. Our method operates in a black-box way and does not require any internal knowledge of the target classifier, which makes our method applicable for more adversarial tasks in cybersecurity. Our method avoids the unstable training of the GAN-based attack methods and has complete control over the generating process of AEs. Our method can perturb the specified features with a specified strength,

which is exactly the deficiency of the GAN-based attack methods.

V. EXPERIMENTS AND ANALYSIS

To have a comprehensive evaluation of the attack effect of our method, we adopt BFAM to generate AEs on three different datasets, i.e., ADFA-LD, NSL-KDD, and DREBIN. These datasets are chosen to validate the performance of our method because host intrusion detection, network intrusion detection, and Android malware detection are the common scenarios in cybersecurity where machine learning algorithms are widely applied. Besides, these datasets can represent the typical inputs for machine learning classifiers in cybersecurity. The machine learning algorithms which are widely used in cybersecurity [1]–[3], [16]–[23] are employed to build up the target detection systems, i.e., LR, DT, MLP, NB, and RF. We regard these three scenarios as binary classification problems. All the normal examples are labeled as 0 and all the malicious examples are labeled as 1.

In cybersecurity, the black-box attack methods are normally thought to be more practical than the white-box attacks [9], [11], [14], [15] because the adversaries can only access the target systems as a standard user most of the time, which means that the adversaries only possess limited knowledge of the target classifiers. Normally, they can just access the output of the target classifiers. Therefore, we only consider the generation of AEs under the setting of black-box attacks in this paper. GAN-based adversarial attack methods are currently one of the most effective and concerned black-box attacks in cybersecurity [9], [14], [15], [31]. The previous work [9] has already compared the attack performance of the common black-box attacks in cybersecurity such as training a substitute model, zeroth-order optimization methods, and GAN-based attack methods. The GAN-based attack methods show excellent performance among these black-box attacks. Besides, our method is proposed to cover some drawbacks of the GAN-based attack methods such as the unstable training *et al.* Therefore, We only compare the attack effect of our method with that of the state-of-the-art GAN-based attack methods. Because the existing GAN-based attack methods [9], [14], [15] all employ the architecture in Fig. 2 to generate AEs, we also adopt this architecture to build up the adversarial attack method based on GAN (AAM-GAN). The Wasserstein GAN (WGAN) [32] shows excellent performance among the various variants of GAN. Therefore, we adopt WGAN to implement the architecture in Fig. 2.

In the real world, the adversaries are more likely to disguise malicious behavior as normal. Therefore, we only consider generating adversarial malicious examples that can evade the detection of the target security systems without changing the functionality of the malicious examples in this paper. The targeted version of BFAM is employed to generate AEs. Besides, we assume that the adversarial attacks happen in the test stage, which is closer to the actual condition. Accordingly, the test data are utilized to produce AEs.

Fig. 3 is adopted to illustrate the generation process of targeted BFAM more intuitively. For an input vector with n features, assuming that the features allowed to be modified are the first six features, BFAM successively modifies the modifiable features to generate AEs and processes a single feature in each iteration. As shown in Fig. 3, the current feature to be perturbed is f_2. BFAM modifies the current feature with specific perturbation strength and recalculates the confidence scores of the target classifiers for the modified input. If the modification increases the confidence score of the target class, the modification is kept. Otherwise, it is canceled and the modified feature is restored to its original value. We then test whether the target classifiers output the target label. If the target classifiers output the target label, we terminate the generation because the adversarial examples have been produced successfully. Otherwise, we continue to modify the next feature f_3. The above generation process proceeds iteratively until the adversarial examples are produced or all the modifiable features are tried. We ignore the clipping procedure of Algorithm 1 in Fig. 3. To generate m AEs with n modifiable features, the computational complexity of the proposed method can be represented as $O(m \times n)$ if BFAM produces m AEs separately. If the vectorized programming is adopted, the computational complexity can be reduced to $O(n)$.

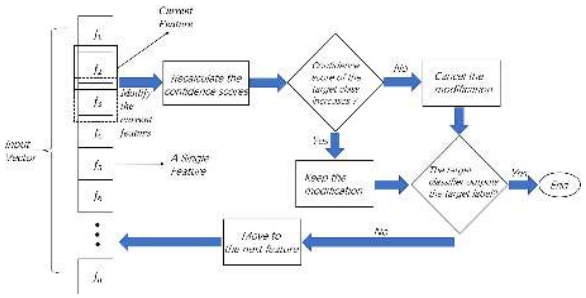

FIGURE 3. The process of generating AEs using targeted BFAM.

The metrics used in this paper include the original detection rate (ODR), adversarial detection rate (ADR), and the total time cost (TTC) of generating AEs. The ODR indicates the detection performance of the target classifiers against the original attack examples. The ODR can be formalized as (6). The ADR indicates the detection performance of the target classifiers against the adversarial attack examples. The ADR can be calculated with (7). Detection rate is a very important metric for machine learning classifiers in cybersecurity. A lower detection rate usually means that many attacks evade the detection of the target detection systems. This will put the systems protected by the target detection systems in danger, which is usually unacceptable. We can intuitively observe the attack effect of the adversarial attack methods and the robustness of the target classifiers against AEs by comparing ODR of the target classifier with its ADR. The attack method shows an excellent attack performance if the target classifier with a high ODR achieves a low ADR on

the adversarial attack examples. Our method can be regarded as an exhaustive but greedy search algorithm. The computational efficiency of our method needs to be concerned. TTC measures the total time consumption of producing a set of AEs. To verify the computational efficiency of BFAM, we compare the TTC of BFAM with that of AAM-GAN in the experiments.

All the experiments are performed on a computer with an i5-8265U CPU and an 8GB RAM. Two 6G Nvidia GTX 1660 GPUs are used to accelerate the computation. We implement all the machine learning classifiers based on Scikit-learn [33] which is a widely used machine learning framework. The deep learning framework Pytorch [34] is used to implement BFAM and AAM-GAN.

$$ODR = \frac{Num.\ of\ correctly\ detected\ original\ attack\ examples}{Num.\ of\ all\ the\ original\ attack\ examples}. \tag{6}$$

$$ADR = \frac{Num.\ of\ correctly\ detected\ adversarial\ attack\ examples}{Num.\ of\ all\ the\ adversarial\ attack\ examples}. \tag{7}$$

A. CRAFTING ADVERSARIAL EXAMPLES AGAINST MACHINE LEARNING BASED HIDS

The original inputs for HIDSs are normally system call traces as discussed in Section III.D. The machine learning classifiers can only process vectors. Therefore, all the original traces need to be transformed into vectors before feeding them to the target classifiers. We adopt the set of words technique to preprocess the original traces because it is widely applied in cybersecurity [23], [35], [36]. Assuming that $\boldsymbol{C} = \{c_1, c_2, c_3, \ldots, c_m\}$ denotes the set of system calls, m is the number of system calls and c_i denotes a single system call. For any system call trace $\boldsymbol{s}$, we transform it into a vector $\boldsymbol{x} =< x_1, x_2, x_3, \ldots, x_m >$, where $x_i = 1$ if $\boldsymbol{s}$ includes c_i else $x_i = 0$. After the preprocessing, all the original system call traces are transformed into vectors with 175 features.

All the malicious system call traces are labeled as 1 and all the normal system call traces are labeled as 0. Thirty percent of the dataset is used as the test set and the rest is used as the training set. The AAM-GAN and the target classifiers share the same training data, which usually means a better performance of the GAN-based attack methods [14]. We adopt the 3-fold cross-validation and grid search to find the optimal parameters for the target classifiers. To guarantee the functionality of the malicious system call traces, we only add system calls to the original malicious system call traces. In this case, ***p_index*** of BFAM is set to the indexes of the features whose values are zero for each input and α is set to 1. The number of training epochs for AAM-GAN is set to 50. The learning rates of the discriminator and the generator are both set to 0.0001. The RMSProp optimizer is adopted to train the discriminator and the generator. The outputs of the generator are firstly clipped into the range of [0, 1]. Then, all the features which are greater than 0.5 are set to 1. All the features which are less than or equal to 0.5 are set to 0. To make sure that we only add additional system calls to original system call traces, we achieve the final AEs generated by AAM-GAN through (8), where $\boldsymbol{x}$ is the original input, $A(\boldsymbol{x})$ denotes the output of AAM-GAN, $\boldsymbol{x}^*$ is the final AE, and | denotes the element-wise OR operation [14].

$$\boldsymbol{x}^* = \boldsymbol{x} | A(\boldsymbol{x}). \tag{8}$$

Table 4 shows the attack performance of BFAM and AAM-GAN against the machine learning based HIDSs. The results show that our method shows an excellent attack performance on all machine learning based classifiers for host intrusion detection. Before the adversarial attacks, all the target classifiers achieve a high detection rate on the original malicious system call traces. However, after transforming the original malicious system call traces into the adversarial malicious system call traces using BFAM, all the adversarial malicious traces evade the detection of the target classifiers. Our method achieves the attack performance comparable to AAM-GAN on LR-based HIDS, NB-based HIDS, MLP-based HIDS, and RF-based HIDS. Our method outperforms AAM-GAN on NB-based HIDS.

TABLE 4. Attacks against machine learning based HIDSs.

Target Classifiers	ODR (%)	ADR (BFAM) (%)	ADR (AAM-GAN) (%)
LR-based HIDS	90.52	0.00	0.00
DT-based HIDS	88.36	0.00	0.00
MLP-based HIDS	83.19	0.00	0.00
NB-based HIDS	81.03	0.00	9.05
RF-based HIDS	82.76	0.00	0.00

Table 5 shows the computational efficiency of BFAM against machine learning based HIDSs. The TTCs in Table 5 indicate the total time consumption of the corresponding attack methods on the whole test set. The TTCs of AAM-GAN in Table 5 include training cost and generation cost. Because BFAM does not need training, the TTCs of BFAM just involves generation cost. As shown in Table 5, the TTCs consumed by BFAM to generate AEs against all the target classifiers are much less than those consumed by AAM-GAN. This proves the effectiveness of BFAM. The TTCs of BFAM and AAM-GAN against RF-based HIDS are much more than the time cost against the other classifiers, which partly shows that ensemble models are more robust

TABLE 5. Computational efficiency of BFAM against machine learning based HIDSs.

Target Classifiers	TTC of BFAM (s)	TTC of AAM-GAN (s)
LR-based HIDS	0.02	22.63
DT-based HIDS	0.05	20.59
MLP-based HIDS	0.16	34.90
NB-based HIDS	0.05	21.89
RF-based HIDS	14.10	150.90

against AEs and the efficiency of BFAM against ensemble models needs to be improved.

B. CRAFTING ADVERSARIAL EXAMPLES AGAINST MACHINE LEARNING BASED AMDS

DREBIN dataset contains 123453 benign Android applications and 5560 malicious Android applications. Arp *et al.* [27] transform every application into a binary indicator vector whose component indicates whether the corresponding feature exists in the target application. If the feature exists in the target application, the corresponding component of the input vector is set to 1. Otherwise, the component is set to 0 [8]. Each application will be transformed into a vector with 54533 features through this method. All the features are grouped into 8 feature sets as shown in Table 3.

We randomly pick out 45000 benign applications as normal samples. All the 5560 malicious applications are used as malicious samples. The benign applications are labeled as 0 and the malicious applications are labeled as 1. Thirty percent of all the selected samples are used as the test set and the others are used as the training set. The AAM-GAN still shares the training set with the target classifiers to achieve a better attack performance. The 10-fold cross-validation and grid search are employed to search the optimal parameters for the target classifiers. To ensure the functionality of the malicious applications, we only modify the nonfunctional features of the applications. Grosse *et al.* [8] have shown that modifying the features in S_1, S_2, S_3, and S_4 does not influence the functionality of malicious applications. Therefore, ***p_index*** includes the indexes of features belonging to these four feature sets. α is still set to 1 because the components of the input vectors are either 0 or 1. We still train the AAM-GAN with the RMSProp optimizer and the learning rate is still set to 0.0001. All the outputs of the generator are clipped into [0, 1]. Then, all the features which are greater than 0.5 are set to 1 and the features which are less than or equal to 0.5 are set to 0. We still employ (8) for AAM-GAN to guarantee that we only add additional features to original malicious applications.

Table 6 shows the attack performance of BFAM and AAM-GAN against the machine learning based AMDSs. BFAM shows excellent attack performance against LR-based, MLP-based, and NB-based AMDSs, whose ADRs against AEs are decreased to 0. This means that all the adversarial malicious applications evade the detection of these target classifiers with their functionality unchanged. BFAM obtains the same performance as AAM-GAN on LR-based and MLP-based AMDSs. BFAM outperforms AAM-GAN on DT-based, NB-based, and RF-based AMDSs. The results in Table 6 also show that RF classifiers are more robust against AEs. Especially, there are still 43.08 percent of adversarial malicious applications generated by AAM-GAN being detected by RF-based AMDS. Although the attack performance of BFAM on RF classifier is worse than its attack performance on the other classifiers, the ADR of RF-based AMDS against adversarial malicious applications generated by BFAM is 17.25 percent, which is far less than 43.08 percent of AAM-GAN.

TABLE 6. Attacks against machine learning based AMDSs.

Target Classifiers	ODR (%)	ADR (BFAM) (%)	ADR (AAM-GAN) (%)
LR-based AMDS	92.73	0.00	0.00
DT-based AMDS	94.21	3.76	9.86
MLP-based AMDS	95.26	0.00	0.00
NB-based AMDS	86.08	0.00	15.23
RF-based AMDS	90.08	17.25	43.08

The computational efficiency of BFAM against machine learning based AMDSs is shown in Table 7. BFAM consumes much less time to generate AEs against LR-based, DT-based, MLP-based, and NB-based AMDSs than AAM-GAN. The TTCs of BFAM and AAM-GAN against RF-based AMDS are still more than their TTCs against the other classifiers. However, the TTC of BFAM against RF-based AMDS is just a quarter of that of AAM-GAN. The results show that our method is more efficient than the GAN-based attack method in the setting of Android malware detection.

TABLE 7. Computational efficiency of BFAM against machine learning based AMDSs.

Target Classifiers	TTC of BFAM (s)	TTC of AAM-GAN (s)
LR-based AMDS	4.17	895.34
DT-based AMDS	3.65	823.86
MLP-based AMDS	25.52	1216.86
NB-based AMDS	2.93	900.83
RF-based AMDS	226.06	1014.87

C. CRAFTING ADVERSARIAL EXAMPLES AGAINST MACHINE LEARNING BASED NIDS

NSL-KDD dataset is a benchmark dataset for network intrusion detection. There are 41 features for each record in the NSL-KDD dataset, which consist of 32 numeric features, 6 binary features, and 3 nominal features. Because machine learning classifiers can only process numeric values, the original records need to be preprocessed before feeding them to machine learning classifiers. Normally, the preprocessing includes two steps [2], [10], [15]: 1) Firstly, the 3 nominal features, i.e., ''protocol type'', ''service'', ''flag'', are transformed into discrete values. For example, the feature, ''protocol type'', has three types of values: TCP, UDP, and ICMP, which will be transformed into 1, 2, 3 correspondingly. 2) Secondly, all the features are scaled into [0,1] with the Min-Max method to avoid the influence of different feature ranges. The Min-Max method can be formalized as (9) where x is the feature value before normalization, x' is the feature value after normalization, x_{min} is the minimum value of this feature over the whole dataset, and x_{max} is the maximum value of this feature over the whole dataset.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{9}$$

There are 4 kinds of malicious traffic data in NSL-KDD. Because the network intrusion detection is regarded as binary classification in this paper, all the malicious traffic data are labeled as 1 and all the normal traffic data are labeled as 0. The NSL-KDD dataset contains training data and test data. We train the target classifiers and AAM-GAN with the whole training set. The 10-fold cross-validation and grid search are utilized to find the target classifiers which show the best detection performance. The whole test set is used to validate the detection performance of the target classifiers and generate AEs against the target classifiers. We still only modify the nonfunctional features in a traffic record. Table 8 [37] shows the functional features of each kind of malicious traffic. The ''Yes'' in Table 8 indicates that the features in the corresponding feature set are functional for that attack category. Because the functional features of each attack category are different, we generate AEs of each attack category separately. Therefore, ***p_index*** contains the indexes of nonfunctional features of the corresponding attack category when BFAM generates AEs of each attack category. In Section V.A and V.B, the inputs for the target classifiers are all binary indicator vectors whose components are either 0 or 1. The α of BFAM can only be set to 1 in the first two scenarios. In the context of NIDS, the input vectors after normalization are the combination of features in continuous values and features in binary values. All the features are in the range of [0,1]. Therefore, the α can be set to any value between 0 and 1 for the continuous features in this setting. We observe the impact of α on the attack performance of BFAM, which is shown in Fig. 4. For the binary features of input vectors, we transform the modified features whose values are greater than 0.5 into 1 and the modified features whose values are less than or equal to 0.5 into 0. With the increase of α, ADRs of LR-based NIDS against AEs of each attack category gradually decrease. The same phenomenon can be observed on the other machine learning based NIDSs. Therefore, we still set α to 1 for BFAM to generate AEs in the following experiments. We adopt the same parameters as those in Section V.A and V.B to train the AAM-GAN. Because the ranges of functional features of different attack categories are different, we train different AAM-GANs for different kinds of attack data on the same target classifier to better evaluate the attack performance of AAM-GAN.

TABLE 8. Functional features of each attack category in the NSL-KDD dataset.

Attack Category	Feature Sets			
	Intrinsic	Content	Time-based Traffic	Host-based Traffic
DoS	Yes	No	Yes	No
Probe	Yes	No	Yes	Yes
U2R	Yes	Yes	No	No
R2L	Yes	Yes	No	No

The attack performance of BFAM and AAM-GAN against machine learning based NIDSs are shown in Table 9. Because the number of samples of U2R is small and samples of U2R and R2L own the same functional features and similar characteristics, we combine them into one group. Due to the small number of samples of U2R and R2L in the training set, the original detection rate of the target classifiers against them is low. Every row in Table 9 shows the original detection rate of a certain kind of target classifier against a certain kind of attack and the adversarial detection rate of the target classifier against the corresponding adversarial malicious traffic generated by BFAM and AAM-GAN. We can observe that different categories of adversarial malicious traffic generated by AAM-GAN perform differently on the same target classifier. For instance, the ADR of NB-based NIDS against adversarial malicious traffic of DoS generated by AAM-GAN is 33.86 percent. However, the ADR of NB-based NIDS against adversarial malicious traffic of Probe generated by AAM-GAN is just 4.88 percent. The similar phenomena can be observed on DT-based, and MLP-based NIDSs. This proves the instability of GAN-based attack methods. The results in Table 9 show that BFAM reduces the ADRs of all the target classifiers greatly and shows better stability than AAM-GAN. BFAM outperforms AAM-GAN in most cases. However, AAM-GAN shows better performance than BFAM when they generate AEs of Probe against LR-based and NB-based NIDSs. RF still shows better robustness against AEs.

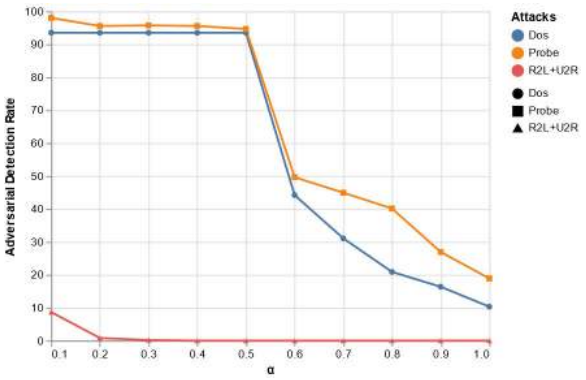

FIGURE 4. Impact of α on the attack performance of BFAM against LR-based NIDS.

The computational efficiency of BFAM against machine learning based NIDSs is shown in Table 10. Just like the results in Table 5 and Table 7, BFAM consumes much less time than AAM-GAN to produce the same number of AEs. The TTCs of BFAM and AAM-GAN on RF are still much more than their TTCs on the other target classifiers.

D. THE IMPACT OF THE NUMBER OF FEATURES ALLOWED TO BE MODIFIED ON ATTACK PERFORMANCE

In this section, we want to discuss the influence of the number of features permitted to be modified in input vectors on the attack performance of BFAM and AAM-GAN. As discussed in Section III.A, the constraint of imperceptibility on AEs in computer vision is substituted by the limitation of guaranteeing the functionality of the adversarial examples in

TABLE 9. Attacks against machine learning based NIDSs.

Target Classifiers	Attack Category	ODR (%)	ADR (BFAM) (%)	ADR (AAM-GAN) (%)
LR-based NIDS	DoS	93.52	10.31	17.84
	Probe	98.19	18.90	12.84
	R2L & U2R	16.55	0.00	0.00
DT-based NIDS	DoS	99.69	12.44	13.46
	Probe	99.55	24.59	43.76
	R2L & U2R	17.26	12.61	13.80
MLP-based NIDS	DoS	98.90	6.31	7.26
	Probe	88.16	18.54	50.81
	R2L & U2R	9.88	0.13	4.96
NB-based NIDS	DoS	81.88	17.56	33.86
	Probe	86.80	15.82	4.88
	R2L & U2R	1.03	0.31	0.86
RF-based NIDS	DoS	99.74	25.85	42.57
	Probe	99.55	36.08	39.42
	R2L & U2R	4.87	0.00	0.00

TABLE 10. The computational efficiency of BFAM against machine learning based NIDSs.

Target Classifiers	Attack Category	TTC (BFAM) (s)	TTC (AAM-GAN) (s)
LR-based NIDS	DoS	0.14	46.60
	Probe	0.03	46.80
	R2L & U2R	0.01	45.80
DT-based NIDS	DoS	0.12	41.72
	Probe	0.03	40.88
	R2L & U2R	0.02	41.74
MLP-based NIDS	DoS	2.86	109.98
	Probe	0.48	112.72
	R2L & U2R	0.14	116.65
NB-based NIDS	DoS	0.16	44.84
	Probe	0.03	45.02
	R2L & U2R	0.02	44.79
RF-based NIDS	DoS	8.19	389.44
	Probe	8.17	387.78
	R2L & U2R	3.44	382.97

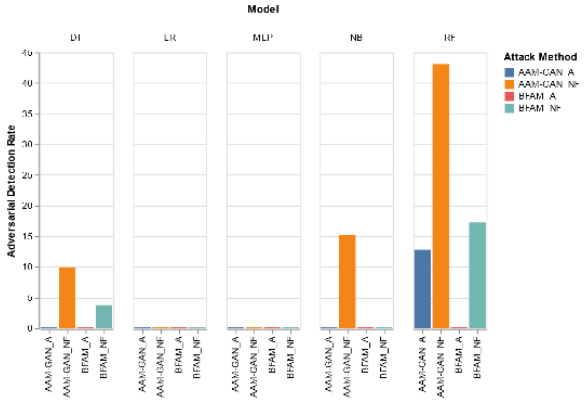

FIGURE 5. Impact of the number of modifiable features on the attack performance against machine learning based AMDSs.

cybersecurity. This means that only nonfunctional features of input vectors can be modified, which usually increases the difficulty of generating AEs. We evaluate the attack performance of BFAM and AAM-GAN against the target classifiers when different numbers of modifiable features are available. Specifically, we specify different numbers of features allowed to be modified for BFAM and AAM-GAN to observe the changes in their attack performance.

The attack performance of BFAM and AAM-GAN against machine learning based AMDSs with different numbers of features allowed to be modified is shown in Fig. 5. The BFAM_A and AAM-GAN_A in Fig. 5 indicate that BFAM and AAM-GAN generate AEs by modifying both functional features and nonfunctional features. BFAM_NF and AAM-GAN_NF indicate that AEs are produced by only altering the nonfunctional features of the input vectors. The results in Fig. 5 indicate that BFAM and AAM-GAN show better attack performance when there is no restriction on the number of modifiable features. The ADRs of DT-based, NB-based, and RF-based AMDSs increase with the decrease in the number of modifiable features, which implies the degradation of attack performance of adversarial attack methods. We can also observe that the performance degradation of BFAM is slower than that of AAM-GAN. For instance, the ADR of the RF-based AMDS against AEs generated by BFAM increases from 0 percent to 17.25 percent after limiting the number of modifiable features. The increase of the ADR against AEs generated by AAM-GAN is from 12.8 percent to 43.08 percent. The ADRs of LR-based and MLP-based AMDSs remain unchanged at 0 percent after restricting the number of modifiable features, which implies that not all the features are useful for generating AEs. We just need to find the key features which can influence the output of the target classifier and modify them appropriately to produce AEs, which proves the point made in Section IV.

The influence of the number of features allowed to be modified on the attack performance against machine learning based NIDSs is shown in Fig. 6. We evaluate the impact of the number of modifiable features on different attack categories in the NSL-KDD dataset separately because attack data of

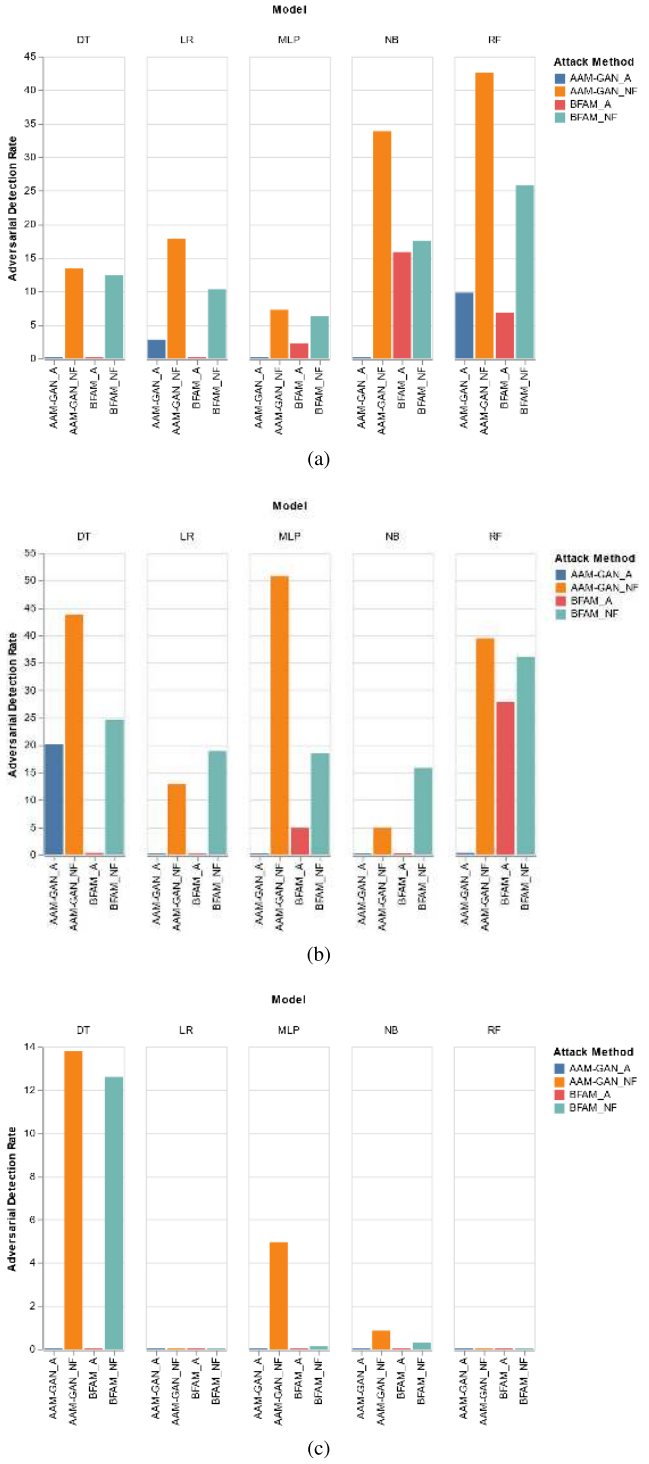

FIGURE 6. Impact of the number of modifiable features on the attack performance against machine learning based NIDSs. (a) displays the impact on the attack performance of AEs of DoS; (b) displays the impact on the attack performance of AEs of Probe; (c) displays the impact on the attack performance of AEs of R2L&U2R.

different categories own different functional features. We still put R2L and U2R into one group. We can draw the same conclusion from Fig. 6. The attack methods show better attack performance when there is no restriction on the number of modifiable features. The degradation of the attack performance of BFAM is slower than that of AAM-GAN when there is a constraint on the number of modifiable features. Besides, we can observe that the performance of AEs of some attack categories generated by AAM-GAN on some target classifiers is better than that of AEs generated by BFAM when there is no limitation on the number of features permitted to be modified. For example, the results in Fig. 6(a) show that AEs produced by AAM-GAN_A achieve a lower ADR than those produced by BFAM_A on MLP-based and NB-based NIDSs.

The results in these figures prove that BFAM is more robust and effective than AAM-GAN when there is a restriction on the number of modifiable features, which is important for the adversarial attack methods in cybersecurity. The impact of the number of modifiable features on the attack performance against machine learning based HIDSs is not displayed in this section because the ADR of machine learning based HIDSs against AEs remains the same even when there is a restriction on the number of modifiable features. This is because we only add system calls into the original system call traces to guarantee the functionality of the inputs. The nonfunctional features of input vectors for machine learning based HIDSs are the ones whose values are 0 in this setting. However, BFAM and AAM-GAN clip the modified features into the range of [0,1] during the generation. AEs that are generated with all the features allowed to be altered and AEs that are generated with just nonfunctional features allowed to be altered obtain the same attack performance.

E. COMPREHENSIVE ANALYSIS

The following analysis and conclusion are given based on the results of the preliminary experiments above:

1) BFAM is simple to implement and avoids the tedious training of GAN-based methods, which makes BFAM more efficient in computation. The results in Table 5, Table 7, and Table 10 prove that BFAM costs much less time to generate AEs than AAM-GAN in various scenarios of cybersecurity.
2) BFAM outperforms the GAN-based attack method in most cases. Especially, when there is a restriction on the number of features allowed to be modified, BFAM is more effective and robust than AAM-GAN.
3) BFAM shows excellent attack performance against state-of-the-art machine learning classifiers in cybersecurity. Most AEs generated by BFAM can evade the detection of the target machine learning based systems in cybersecurity, which means state-of-the-art machine learning algorithms are vulnerable to AEs. Among all the target classifiers, RF classifiers are the most robust against AEs.
4) BFAM is a black-box attack method, which only requires the confidence scores of the target classifiers to generate AEs. This makes BFAM closer to the real condition and suitable for more adversarial tasks in cybersecurity than the white-box attacks.

5) Currently, the training of the GAN-based attack methods is unstable which is reflected in the instability of the attack performance of AAM-GAN. For instance, the ADR of MLP-based NIDS against AEs of Probe generated by AAM-GAN is 50.81 percent in Table 9. Nevertheless, the ADR of NB-based NIDS against AEs of Probe generated by AAM-GAN is just 4.88 percent. The GAN-based methods show better attack performance when there is no restriction on the number of modifiable features. For example, when all the features can be modified, AAM-GAN decreases the ADRs of LR-based, MLP-based, and NB-based NIDSs against AEs of Probe to 0 percent, as shown in Fig. 6(b). However, we need to keep the functional features unmodified to guarantee the validity of the AEs in the real world, which hinders the further application of the existing GAN-based attack methods.
6) GAN-based attack methods can produce AEs only with labels. BFAM needs the confidence scores outputted by the target classifiers to instruct the generation of AEs, which means that the proposed method is currently not fit for being used to evaluate the machine learning classifiers which only output labels. BFAM requires to be improved to be applicable for more machine learning classifiers in future work.
7) Comparing the results in Table 4, Table 6, and Table 9, we can conclude that BFAM performs better on machine learning based HIDSs and AMDSs whose inputs are high dimensional sparse vectors. The intuitive hypothesis for this is that BFAM is an exhaustive algorithm for which more features allowed to be modified means more opportunities to mislead the target classifiers.

VI. CONCLUSIONS

The preliminary experimental results in this paper indicate that the proposed method, BFAM, shows excellent attack performance against the common machine learning algorithms utilized in cybersecurity. Therefore, BFAM can be used to evaluate the robustness of machine learning based systems in cybersecurity against AEs. BFAM outperforms the state-of-the-art GAN-based attack methods and produces AEs more simply and efficiently. BFAM decreases the detection rate of the target classifiers against adversarial malicious examples greatly without changing the functionality of these malicious examples. This usually means that most of the adversarial malicious examples evade the detection of the target classifiers, which is unacceptable for the security-critical systems in cybersecurity. BFAM operates in a black-box way and only requires the confidence scores of the target classifiers to generate AEs, which makes BFAM available for more adversarial attack tasks in cybersecurity.

Sometimes, the adversaries can only access the labels predicted by the target classifier. BFAM is not able to work in this case, as discussed in Section V.E. In future work, we are going to improve BFAM so that it can generate AEs only with the labels and can be used for more scenarios and target classifiers.

REFERENCES

[1] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.
[2] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2nd Quart., 2016.
[3] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.
[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*. [Online]. Available: http://arxiv.org/abs/1312.6199
[5] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 372–387.
[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2574–2582.
[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.
[8] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2017, pp. 62–79.
[9] K. Yang, J. Liu, C. Zhang, and Y. Fang, "Adversarial examples against the deep learning based network intrusion detection systems," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 559–564.
[10] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38367–38384, 2018.
[11] X. Liu, X. Du, X. Zhang, Q. Zhu, H. Wang, and M. Guizani, "Adversarial samples on Android malware detection systems for IoT systems," *Sensors*, vol. 19, no. 4, p. 974, Feb. 2019.
[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[13] Y. Hong, U. Hwang, J. Yoo, and S. Yoon, "How generative adversarial networks and their variants work: An overview," *ACM Comput. Surv.*, vol. 52, no. 1, p. 10, 2019.
[14] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, *arXiv:1702.05983*. [Online]. Available: http://arxiv.org/abs/1702.05983
[15] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," 2018, *arXiv:1809.02077*. [Online]. Available: http://arxiv.org/abs/1809.02077
[16] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.
[17] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 453–466, Feb. 2019.
[18] N. M. Shajideen and V. Bindu, "Spam filtering: A comparison between different machine learning classifiers," in *Proc. 2nd Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Coimbatore, India, Mar. 2018, pp. 1919–1922.
[19] A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, "Android malware detection using genetic algorithm based optimized feature selection and machine learning," in *Proc. 42nd Int. Conf. Telecommun. Signal Process. (TSP)*, Budapest, Hungary, Jul. 2019, pp. 220–223.
[20] H. Yao, D. Fu, P. Zhang, M. Li, and Y. Liu, "MSML: A novel multilevel semi-supervised machine learning framework for intrusion detection system," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1949–1959, Apr. 2019.
[21] J. Ren, J. Guo, W. Qian, H. Yuan, X. Hao, and H. Jingjing, "Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms," *Secur. Commun. Netw.*, vol. 2019, Jun. 2019, Art. no. 7130868, doi: 10.1155/2019/7130868.
[22] D. S. Vijayakumar and S. Ganapathy, "Machine learning approach to combat false alarms in wireless intrusion detection system," *Comput. Inf. Sci.*, vol. 11, no. 3, pp. 67–81, 2018.

[23] E. Sahin, M. Aydos, and F. Orhan, "Spam/ham e-mail classification using machine learning methods based on bag of words technique," in *Proc. 26th Signal Process. Commun. Appl. Conf. (SIU)*, zmir, Turkey, May 2018, pp. 1–4.
[24] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.
[25] A. Chawla, B. Lee, S. Fallon, and P. Jacob, "Host based intrusion detection system with combined CNN/RNN model," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2018, pp. 149–158.
[26] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2013, pp. 4487–4492.
[27] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[28] R. Taheri, R. Javidan, M. Shojafar, P. Vinod, and M. Conti, "Can machine learning model with static features be fooled: An adversarial machine learning approach," *Cluster Comput.*, Mar. 2020, doi: 10.1007/s10586-020-03083-5.
[29] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *Proc. 26th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 2018, pp. 533–537.
[30] F. Zhang, P. P. K. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 766–777, Mar. 2016.
[31] A. Piplai, S. S. L. Chukkapalli, and A. Joshi, "NAttack! Adversarial attacks to bypass a GAN based classifier trained to detect network intrusion," 2020, *arXiv:2002.08527*. [Online]. Available: http://arxiv.org/abs/2002.08527
[32] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, *arXiv:1701.07875*. [Online]. Available: http://arxiv.org/abs/1701.07875
[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, and D. Cournapeau, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[34] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Proc. NIPS Autodiff Workshop, Future Gradient-Based Mach. Learn. Softw. Tech-Niques.*, 2017, pp. 1–4.
[35] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2015, pp. 11–20.
[36] 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 Gener. Comput. Syst.*, vol. 105, pp. 230–247, Apr. 2020.
[37] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 227–261, Nov. 2000.

SICONG ZHANG was born in Chongqing, China. He received the B.E. degree in electrical engineering and automation from the Civil Aviation University of China and the M.E. degree in computer science and technology from Guizhou Normal University. He is currently pursuing the Ph.D. degree in software engineering with Guizhou University, Guiyang, China. His research interests include cybersecurity, deep learning, and optimization theory.

XIAOYAO XIE (Member, IEEE) was born in Guizhou, China. He is currently a Professor and a Ph.D. Supervisor with the Key Laboratory of Information and Computing Science Guizhou Province, Guizhou Normal University, Guiyang, China. He is also the Director of the Key Laboratory. He is also the Vice President of Guizhou Normal University. His research interests include IPv6, 5G, and cybersecurity.

YANG XU was born in Shandong, China. He received the Ph.D. degree in computer software and theory from Guizhou University. He is currently a Professor and a master's Supervisor with the Key Laboratory of Information and Computing Science Guizhou Province, Guizhou Normal University, Guiyang, China. His research interests include cybersecurity and machine learning. He is a Senior Member of the China Computer Federation (CCF).