N-gram MalGAN: Evading machine learning detection via feature n-gram

Enmin Zhu, Jianjie Zhang, Jijie Yan, Kongyang Chen, Chongzhi Gao

Abstract

In recent years, many adversarial malware examples with different feature strategies, especially GAN and its variants, have been introduced to handle the security threats, e.g., evading the detection of machine learning detectors. However, these solutions still suffer from problems of complicated deployment or long running time. In this paper, we propose an n-gram MalGAN method to solve these problems. We borrow the idea of n-gram from the Natural Language Processing (NLP) area to expand feature sources for adversarial malware examples in MalGAN. Generally, the n-gram MalGAN obtains the feature vector directly from the hexadecimal bytecodes of the executable file. It can be implemented easily and conveniently with a simple program language (e.g., C). The features are functionally independent and thus can be added to the non-functional area of the malicious program to maintain its original executability. In this way, the n-gram could make the adversarial attack easier and more convenient. Experimental results show that the evasion rate of the n-gram MalGAN is at least 88.58% to attack different machine learning algorithms under an appropriate group rate, growing to even 100% for the Random Forest algorithm.

1. Introduction

With the rapid development of the information industry, more and more security threats occur in existing information systems, for instance, the malware problem in cybersecurity areas. Due to their high accuracy and adaptability, machine learning solutions have become a vital tool to handle such network security problems or detect potential threats in many executable tasks. However, researchers found that these solutions can also be easily cheated. For example, if the attacker adds several specific subtle perturbations into the original software, these machine learning solutions will achieve wrong results with high confidence. This is the so-called adversarial malware example.

A typical adversarial malware example is to reform some key features of the existing malware such as byte sequences, APIs, opcodes, and strings [1–4]. More specifically, the attacker will first analyze a large number of malicious software programs to find out some special features which are strong enough to show the characteristics of the original malware. After that, the origin malware can be mapped into a feature vector with these features. Finally, the attacker will be able to modify, add, or delete these features to make the anti-virus engine generate a wrong result, even without affecting the malware’s functions.

In recent years, a series of adversarial malware examples with different feature strategies are introduced. For example, Bojan Kolosnjaji et al. generated the features by a gradient-descent algorithm to modify some specific bytes of malware [5]. Based on the idea of GAN [6], Hu et al. chose the original features from API function calls [7]. Then, they mapped the origin malware to the feature space and obtained some adversarial examples with GAN. However, to extract those features, the attacker must deploy and utilize many external tools, which is very complicated and time-consuming.

To solve this problem, we propose an n-gram model to generate adversarial malware examples. The n-gram is a common language model in NLP, which is often used for speech recognition, handwritten recognition, machine translation, spelling correction, etc. In network security, the n-gram model is also well-known in software feature representation [8]. In this paper, by borrowing the idea of n-gram, we extract the contents of a sample into a long string of hexadecimal bytecodes. For example, two words are treated as 2 g, which can be considered as the feature of training samples in MalGAN. Compared with the existing feature strategies, the n-gram expands feature sources for adversarial

https://doi.org/10.1016/j.dcan.2021.11.007
Received 28 November 2020; Received in revised form 15 October 2021; Accepted 17 November 2021
Available online 25 November 2021
2352-8648/© 2021 Chongqing University of Posts and Telecommunications. Publishing Services by Elsevier B.V. on behalf of KeAi Communications Co. Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
malware examples, and also achieves excellent experimental results in the MalGAN training process.

Generally, we summarize our contributions and key finding as follows:

- We borrow the idea of n-gram to expand feature sources for adversarial malware examples in MalGAN and propose the n-gram MalGAN method, which combines the features from malware samples and benign samples to evade the machine learning models.

- **Easy to extract the features:** N-gram MalGAN obtains the feature vector directly from the hexadecimal bytecodes of the executable file. It can be implemented easily and conveniently with a simple program language (e.g., C++,), with no need for any prior knowledge of the executable file or any professional feature extraction tools.

- **Easy to evade the machine learning model:** The features in the n-gram MalGAN are functionally independent of the executable files. So, they can be added to the non-functional area of the malicious program to maintain its original executability. It makes the adversarial attack easier and more convenient.

- **Fast convergence and more stable:** In the n-gram MalGAN, we do not add noises in the malicious samples in the adversarial generation process, so it becomes more stable with rapid convergence.

- **Wide application scenarios:** Prior methods (e.g., MalGAN) simply assume that the machine learning detection detects the malware with the DLL or API, which is a strong hypothesis for the black detector. In the n-gram MalGAN, we extract the features directly from the executable file rather than the DLL or API of the executable file. Thus, it can also be used in more application scenarios.

- Experimental results show that the evasion rate of the n-gram MalGAN is at least 88.58% to attack different machine learning algorithms, growing to even 100% for the Random Forest (RF) algorithm.

The remainder of this paper is organized as follows. Section 2 introduces the related works. Section 3 states the motivation of the work. Section 4 describes our approach in detail. Section 5 shows the experiment results. Finally, Section 6 concludes this paper.

2. Related works

In this section, we introduce some related works in adversarial malware areas, including malware detection and evasion.

2.1. Malware detection

Nowadays, malware has become more and more complex in form and content. To prevent potential security problems and minimize the threat, malware detection has become a critical issue. Typical malware detection methods include signature monitoring method, behavior monitoring method, heuristic method, behavioral results detection, and machine learning based detection.

2.1.1. Signature monitoring method

This method compares the signatures of the existing malicious samples with the tested samples. If two signatures match, the tested samples are malicious files.

2.1.2. Behaviour monitoring method

This method uses the behavioral characteristics of the malicious software for malware detection.

2.1.3. Heuristic method

The danger level is defined by the call relationships. If the detected software calls a specific set of dangerous functions, it may be malware.

2.1.4. Behavioural results detection

In a secure environment, the software is executed to acquire a system status. By comparing the system status changes, we can judge whether it is infected with a malicious code.

2.1.5. Machine learning based detection

Saxe et al. proposed a method to extract four different types of complementary features from static benign and malicious files [9], such as byte/entropy histogram features, PE import features, String 2d histogram features, and PE metadata features. With these features, they trained a network, and the network output shows the probability whether this software is malware. Wang et al. used the Gan-assisted DNN to construct a more stable malware detector [10]. Raff et al. over-wrote an original binary file of software into CNN directly and then trained a network model to detect malware [11]. Cui et al. transformed the binary code of malware into pictures, which converts the malware detection problem into a classic image classification problem [12]. Wang et al. extracted the high-dimensional features of Android APPs and then detected Android malware with a hybrid model based on deep autoencoder and CNN [13]. Kim et al. presented a malware detection approach with deep transferred GAN to automatically classify malicious software [14].

2.1.6. Intrusion detection

There are also many collaborative intrusion detection methods to recognize the anomaly events with semi-supervised machine learning [15–17] or supervised learning [18–20], which can be adopted in the malware detection area easily.

2.2. Malware evasion

Generative Adversarial Network (GAN) is proposed by Goodfellow [6]. The structure of the model is shown in Fig. 1, which includes two neural networks: a generator and a discriminator. During the training process, the generator and the discriminator continue to fight against each other. The generator continuously learns to generate adversarial samples similar to the training data to evade the detection of the discriminator. The discriminator works to accurately distinguish the training data and the data generated by the generator. The whole train process stops until the discriminator could not identify the adversarial examples generated by the generator.

There are also many variant GAN methods. Radford et al. proposed the Deep Convolution Generative Adversarial Network (DCGAN) [21]. They first turned the software into an image to train the model to generate adversarial networks. After that, they reused part of the network of generators and discriminators as feature extractors for supervised tasks. Chen et al. proposed InfoGAN [22], which extends the information theory of generative adversarial networks and learns unwrapping representations in a completely unsupervised manner. During the training process, they modified the generative adversarial network targets to maximize the mutual information between a fixed subset of GAN noise variables and observations. Hence, it will learn interpretable and meaningful representations. Hu et al. proposed MalGAN to generate anti-malware samples to avoid black-box detection [7]. Specially, the generator will add features to the original features and evade the discriminator. Based on MalGAN, Kawai et al. proposed an improved-MalGAN [23]. During the training process, they used different features and a differentiated learning method with only one malware
3. Motivation

In the cybersecurity areas, there are many security threats such as the well-known malware problem. To handle these security problems, many machine learning approaches have been deployed to detect potential security threats. However, these machine learning approaches might also be cheated if the attacker generates many adversarial malware examples (e.g., only adds several subtle perturbations into the original software).

Existing adversarial malware examples have been generated by introducing some modification to the key features (e.g., byte sequences, APIs, and DLLs) of an original software. Their target is to make the antivirus engine of the machine learning approaches fail to identify the adversarial malware examples even when the original software is previously marked as malware. Following this idea, GAN and its variants have been used to improve the adversarial generation performance.

However, these adversarial malware generation methods still have many drawbacks. For example, these solutions need key features such as byte sequences, APIs, or DLLs of previous malware to generate adversarial malware examples. Thus, they cannot produce a good adversarial malware example if these key features are unknown. In addition, they still need professional feature extraction tools to acquire these key features. Finally, these adversarial malware examples are highly related to these key features, so it is very difficult to add them to the previous malware without affecting its default functionality.

To tackle these problems, we aim to present a new adversarial malware example generation method with easier feature extraction, convenient model evading, wider application scenarios, etc.

4. Our approach

As mentioned above, MalGAN applies the GAN to generate adversarial examples. In our approach, we improve the original feature in MalGAN with the n-gram feature to improve the feature extraction.

4.1. N-gram

Generally, an n-gram is simply a sequence of n words. In the NLP area, we often assign a probability to the occurrence of an n-gram for word prediction. In the cybersecurity area, the n-gram is also very useful in malware detection [8]. For example, each program is constituted of binary codes, and we can consider a sequence of n codes as an n-gram [25]. In an n-gram, we can set different n-values to fit each application scenario, e.g., n = 2 in our method.

4.2. Feature extraction

The purpose of feature extraction is to extract information which characterizes the software for the adversarial examples generation. The feature extraction process is shown in Fig. 2. In our method, we would like to find out many representative features of benign samples, which is then added to malicious samples so that the malicious samples can evade the black-box detection.

We first extract features of malicious samples. Specifically, we count each 2-g in all malicious samples to obtain their numbers and proportions. All 2-g are sorted in descending order, and those 2-g with larger proportions are selected as the feature set that represents malicious samples. After that, we use the same method to extract 2-g from the benign samples as the feature set.

To ensure the two feature sets are not overlapped, we combine them into a feature matrix. In our experiment, we set the number of features in this matrix as \( m + n = 350 \), where \( m \) is the number of features for malware, and \( N \) is the number of features for benign software. Finally, all samples are traversed to obtain the corresponding characteristic matrix. Generally, we map each malware sample and each benign sample to a 350-dimensional vector space. These vectors are later used to train the MalGAN, as discussed below.

4.3. N-gram MalGAN

4.3.1. MalGAN

MalGAN consists of a generator, a substitute detector, and a black-box detector, which is shown in Fig. 3.

In MalGAN, both the generator and the substitute detector are feed-forward neural networks and their loss functions can be described as Formulas (1) and (2):

\[
L_G = E_{\text{Malware log}} D_{\text{Malware}}(G_{\theta_1}(m, z)) \quad \text{(1)}
\]

\[
L_D = -E_{\text{Benign log}} \log(1 - D_{\theta_2}(x)) - E_{\text{Malware log}} \log D_{\theta_2}(x) \quad \text{(2)}
\]

The black-box detector is a machine learning based malware detection algorithm. Its internal parameters are not visible. The attacker can only obtain the detection results generated by a black-box detector. During the training process, the generator and the substitute detector are trained to attack the black-box detector based on the machine learning method. First, the feature vector of the malware sample is altered with added noise and sent to the generator as an input. The original features are maintained in the output of the generator to form an adversarial example. Then, the adversarial example is sent to the black-box detector. Finally, the generator and the substitute detector are optimized with the result of the black-box detector.

In MalGAN, added noise in malicious samples is used in the adversarial generation. However, this noise is uncontrollable, which brings uncertainty and instability to adversarial sample generation. To solve this problem, our method does not add any noise.

4.3.2. Combination of n-gram and MalGAN

Inspired by MalGAN, we use the n-gram model to represent the features of the software. We also add some benign software features to the malware features to realize the generation of adversarial examples. This

![Fig. 2. Feature extraction for adversarial samples generation.](image-url)
is called n-gram MalGAN, and its framework is shown in Fig. 4.

In the n-gram MalGAN, both the generator and the substitute detector are feed-forward neural networks. The black-box detector is an anti-malware system based on machine learning algorithms, such as RF, Support Vector Machine (SVM), etc., whose internal parameters are not visible. The training process of n-gram MalGAN is very similar to that of MalGAN.

In the n-gram MalGAN, the original features of the malware will not be modified to ensure the malicious function of the original software. Specifically, we take the ‘OR’ operation to achieve this goal as following: 

\[
B' = A \text{ OR } B = [1, 0, 1, 1, 0, \ldots, 0, 1] \text{ OR } [0, 0, 1, 1, 0, 1 \ldots, 0, 0] = [1, 0, 1, 1, 1, \ldots, 0, 1].
\]

In Fig. 4, we extract the 350 dimension features to represent malware and feed it into the generator. After that, we contain the first 300 dimensions features unchanged.

5. Experiments

In this section, we evaluate our n-gram MalGAN method with different experiments.

5.1. Experiment setup

Here we introduce our experimental environment, our dataset, and feature extraction methods, etc.

5.1.1. Experimental environment

In this experiment, we use CPU Intel i5-8300H, RAM 8.00 GB, Windows 10 system. The Keras 2.3.1 software is used to construct the n-gram MalGAN. The sklearn 0.21.2 software is used to create the malware detector. The Program language is Python 3.7.2.

5.1.2. Dataset

Malware samples are crawled from Kaggle, Microsoft malware classification challenge (BIG 2015) [26]. The benign samples are the normal PE files in our computer. Specifically, the dataset consists of two parts, including 5000 malicious samples and 4000 benign samples. Among that, 80% of the dataset is regarded as the training set, 20% as the test set.

5.1.3. Feature extraction

In the experiment, we map each sample to a 350-dimensional vector. Specifically, we write a C++ script to extract the n-gram from the binary program files. To reduce the computation cost, we first convert the binary file into a hexadecimal file so that each word can be seen as 1-g. We can construct 2-g with an appropriate cut point. For example, given a certain program “6AEF28DC” in hexadecimal, we can determine two 2-g “6AEFEF28” and “EF2828DC” with an overlapped “EF28”. These 2-g vectors are used as the training input of the MalGAN. The experimental settings of feature combination are also shown in Table 1.

5.1.4. Black-box detector

In our experiment, many machine learning algorithms are considered as the black detector, such as RF, Logistic Regression (LR), Decision Trees (DT), SVM, and Multi-Layer Perceptron (MLP). The overall detection results are obtained with a voting method based on the ensemble of these classifiers.

<table>
<thead>
<tr>
<th>Set</th>
<th>Features [Malware, Benign]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>[300, 50]</td>
</tr>
<tr>
<td>Set 2</td>
<td>[330, 20]</td>
</tr>
<tr>
<td>Set 3</td>
<td>[340, 10]</td>
</tr>
</tbody>
</table>

Table 1

Experimental settings of feature combination.

Fig. 3. The structure of MalGAN.

Fig. 4. The framework of n-gram MalGAN.
5.2. Experimental results

In the experiment, we choose the True Positive Rate (TPR) to reflect how many malware samples successfully bypass the detection model.

5.2.1. Different dataset partition methods

We first evaluate how the dataset partition affects the attack performance. Generally, we use two dataset partition methods. The first one is to use the same dataset in the training process of the black-box detector and the n-gram MalGAN. The second one is to divide the dataset into two parts with equal size, the black-box detector and an n-gram MalGAN having each part separately.

5.2.1.1. Same dataset for the black-box detector and the n-gram MalGAN. As shown in Table 2, for each black-box, TPRs on the original examples are 90% greater than both the training set and the test set, which means we have a good black-box to detect almost all malicious examples. The TPRs on the adversarial examples are reduced to below 11.42%, even 0% for the RF model. In other words, the evasion rate is between 88.58% and 100%. It means that after the training process of the n-gram MalGAN, the black-boxes have a good detection rate on the original examples, which could hardly detect any malware from the generator. The experimental results show that the n-gram MalGAN has a successful learning ability to bypass these machine detectors.

The convergence curve of TPR during the training process of the n-gram MalGAN is shown in Fig. 5. After the first 50 epochs of the training process, TPR grows to the stationary point. It means that our n-gram MalGAN is relatively stable and very fast to learn.

5.2.1.2. Different datasets for the black-box detector and the n-gram MalGAN. We also analyzed another dataset partition method where the black-box and n-gram MalGAN are trained with different datasets. The final dataset partition that gives the best result is [300 malware-representative features and 50 benign-example-representative features].

The training results are shown in Table 3. We can infer that the detection rate for the original sample reduced significantly, especially 88.07% for DT.

5.2.2. Different feature combination methods

In this section, we verified the impact on the generation of adversarial samples by comparing three different feature combinations methods as follows:

- Tables 2 and 3: 350 dimensional feature vectors from [300 malware-representative features, 50 benign-example-representative features].
- Tables 4 and 5: 350 dimensional feature vectors from [330 malware-representative features, 20 benign-example-representative features].
- Tables 6 and 7: 350 dimensional feature vectors from [340 malware-representative features, 10 benign-example-representative features].

The experimental results for these feature combination methods are shown in Tables 2~7. We can see that the detection rate of adversarial examples is higher than that in Table 3, which indicates that the number of adversarial examples is lower than that in Table 3. This proves that feature selection is necessary.

5.2.3. Noise addition

To exploit the feature selection in the n-gram MalGAN, we replaced the benign-example-representative features in the selected feature vectors with random noise. This random noise is randomly selected n-gram features. In this experiment, the feature vectors are [300 malware-representative features, 50 benign-example-representative features]. We ran several experiments using multiple datasets with noise addition. Experimental results are shown in Table 8 and Table 9.

In Tables 8 and 9, the detection rate of the adversarial examples and the original examples are very similar, which means that adversarial examples are hard to escape the detection of the black box. The reason is that the generation range of the generator is too small, which makes the adversarial examples hard to obtain strong benign features.

Table 2

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (%)</td>
<td>Adver (%)</td>
</tr>
<tr>
<td>MLP 98.22</td>
<td>10.58</td>
</tr>
<tr>
<td>DT 91.84</td>
<td>11.27</td>
</tr>
<tr>
<td>LR 99.1</td>
<td>9.26</td>
</tr>
<tr>
<td>SVM 97.53</td>
<td>8.05</td>
</tr>
<tr>
<td>RF 97.02</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>TPRs on different datasets.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
</tr>
<tr>
<td>Original (%)</td>
</tr>
<tr>
<td>MLP 98.3</td>
</tr>
<tr>
<td>DT 88.07</td>
</tr>
<tr>
<td>LR 98.87</td>
</tr>
<tr>
<td>SVM 97.17</td>
</tr>
<tr>
<td>RF 97.12</td>
</tr>
</tbody>
</table>

As shown in Table 5, the results are similar to Table 4, which shows the number of adversarial examples that escapes the black box detection is much lower than that in Table 3.

From Tables 6 and 7, we know that the detection rates of original examples and adversarial examples are very similar, which means that adversarial examples are hard to escape the detection of the black box. The reason is that the generation range of the generator is too small, which makes the adversarial examples hard to obtain strong benign features.
We have compared our n-gram MalGAN with the well-known MalGAN method in several aspects.

### Table 5
TPRs on [330 malware-representative features, 20 benign-example-representative features] and different datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original(%)</td>
<td>Advert(%)</td>
</tr>
<tr>
<td>MLP</td>
<td>97.99</td>
<td>86.1</td>
</tr>
<tr>
<td>DT</td>
<td>93.26</td>
<td>73.24</td>
</tr>
<tr>
<td>LR</td>
<td>98.71</td>
<td>39.63</td>
</tr>
<tr>
<td>SVM</td>
<td>97.02</td>
<td>56.77</td>
</tr>
<tr>
<td>RF</td>
<td>98.25</td>
<td>89.91</td>
</tr>
</tbody>
</table>

### Table 6
TPRs on [340 malware-representative features, 10 benign-example-representative features] and the same dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original(%)</td>
<td>Advert(%)</td>
</tr>
<tr>
<td>MLP</td>
<td>99.05</td>
<td>95.21</td>
</tr>
<tr>
<td>DT</td>
<td>93.65</td>
<td>93.65</td>
</tr>
<tr>
<td>LR</td>
<td>98.74</td>
<td>66.53</td>
</tr>
<tr>
<td>SVM</td>
<td>98.25</td>
<td>97.71</td>
</tr>
<tr>
<td>RF</td>
<td>98.79</td>
<td>97.89</td>
</tr>
</tbody>
</table>

### Table 7
TPRs on [340 malware-representative features, 10 benign-example-representative features] and different datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original(%)</td>
<td>Advert(%)</td>
</tr>
<tr>
<td>MLP</td>
<td>99.28</td>
<td>95.32</td>
</tr>
<tr>
<td>DT</td>
<td>91.15</td>
<td>83.79</td>
</tr>
<tr>
<td>LR</td>
<td>98.66</td>
<td>66.7</td>
</tr>
<tr>
<td>SVM</td>
<td>98.19</td>
<td>97.74</td>
</tr>
<tr>
<td>RF</td>
<td>99.07</td>
<td>94.85</td>
</tr>
</tbody>
</table>

### Table 8
TPRs on [300 malware-representative features, 50 noise features] and the same dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original(%)</td>
<td>Advert(%)</td>
</tr>
<tr>
<td>MLP</td>
<td>98.66</td>
<td>98.66</td>
</tr>
<tr>
<td>DT</td>
<td>92.72</td>
<td>92.72</td>
</tr>
<tr>
<td>LR</td>
<td>98.12</td>
<td>95.88</td>
</tr>
<tr>
<td>SVM</td>
<td>98.94</td>
<td>97.92</td>
</tr>
<tr>
<td>RF</td>
<td>97.27</td>
<td>97.27</td>
</tr>
</tbody>
</table>

### Table 9
TPRs on [300 malware-representative features, 50 noise features] and different datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original(%)</td>
<td>Advert(%)</td>
</tr>
<tr>
<td>MLP</td>
<td>98.51</td>
<td>98.97</td>
</tr>
<tr>
<td>DT</td>
<td>91.92</td>
<td>92.49</td>
</tr>
<tr>
<td>LR</td>
<td>98.35</td>
<td>97.17</td>
</tr>
<tr>
<td>SVM</td>
<td>97.48</td>
<td>98.51</td>
</tr>
<tr>
<td>RF</td>
<td>99.02</td>
<td>98.15</td>
</tr>
</tbody>
</table>

### 5.3. Comparison with MalGAN

In this paper, we extend the feature extraction for adversarial malware examples directly from the hexadecimal bytecodes of the executable file, thus do not need any prior knowledge. Furthermore, these features from the malware and benign samples are collected to form a 350-dimensional feature vector. Our experimental results show that the ratio of benign features plays an important role in the overall evading accuracy. For example, if the ratio of benign features is too large, the n-gram MalGAN will be easily detected by the malware detector. Otherwise, if the ratio of benign features is too small, the n-gram MalGAN fails to learn enough benign features to evade the machine learning detection algorithm. Therefore, we should run the experiment multiple times to find an appropriate ratio of the benign features (e.g., 300 malware-representative features and 50 benign-example-representative features in Tables 2 and 3).

### 5.4. Discussion

In this paper, we extend the feature extraction for adversarial malware examples directly from the hexadecimal bytecodes of the executable file, thus do not need any prior knowledge. Furthermore, these features from the malware and benign samples are collected to form a 350-dimensional feature vector. Our experimental results show that the ratio of benign features plays an important role in the overall evading accuracy. For example, if the ratio of benign features is too large, the n-gram MalGAN will be easily detected by the malware detector. Otherwise, if the ratio of benign features is too small, the n-gram MalGAN fails to learn enough benign features to evade the machine learning detection algorithm. Therefore, we should run the experiment multiple times to find an appropriate ratio of the benign features (e.g., 300 malware-representative features and 50 benign-example-representative features in Tables 2 and 3).
6. Conclusions

In this paper, we use MalGAN to generate adversarial examples. To improve the performance of adversarial examples generation, the n-gram model is also borrowed for feature selection. More specifically, we take the n-gram feature as input to replace the original API feature in MalGAN. We evaluated the feasibility of our method, and the experimental results show that the n-gram is an excellent adversarial feature to make the attack more effective. With an appropriate group rate, the evasion rate is at least 88.58% to attack different machine learning algorithms, growing to even 100% for the RF algorithm.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by Natinal Science Foundation of China (No. 61802383), Research Project of Pazhou Lab for Excellent Young Scholars (No. PZL2021KF0024), Guangzhou Science and Technology Project Basic Research Plan (No. 202201010330, 202201020162), Guangdong Philosophy and Social Science Planning Project (No. GD19YYJ02), Research on the Supporting Technologies of the Metaverse in Cultural Media (No. PT252022039), and National Undergraduate Training Platform for Innovation and Entrepreneurship (No. 202111078029).

References


Digital Communications and Networks 8 (2022) 485–491

491