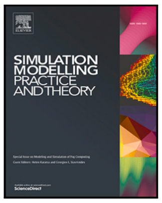
Simulation Modelling Practice and Theory 113 (2021) 102391

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



Malicious mining code detection based on ensemble learning in cloud computing environment

Shudong Li[a,1,*], Yuan Li[a,1], Weihong Han[a], Xiaojiang Du[b,2], Mohsen Guizani[c,2], Zhihong Tian[a,*]

[a] *Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, 510006, China*
[b] *Department of Computer and Information Sciences, Temple University, Philadelphia, USA*
[c] *Computer Science and Engineering Department, Qatar University, Qatar*

ARTICLE INFO

Keywords:
Malicious mining code
Mining virus
Cloud computing
Static analysis
Ensemble learning

ABSTRACT

Hackers increasingly tend to abuse and nefariously use cloud services by injecting malicious mining code. This malicious code can be spread through infrastructures in the cloud platforms and pose a great threat to users and enterprises. In this study, a method is proposed for detecting malicious mining code in the cloud platforms, which constructs a detection model by fusing the Bagging and Boosting algorithms. By randomly extracting samples and letting models vote together to decide, the variance of model detection can be reduced obviously. Compared with traditional classifiers, the proposed method can obtain higher accuracy and better robustness. The experimental results show that, for the given dataset, the values of AUC and F1-score can reach 0.992 and 0.987 respectively, and the standard deviation of AUC values under different data inputs is only 0.0009.

1. Introduction

In the past few years, with the rapid popularity of cloud computing, cloud security has become one of the challenges faced by many cloud service providers and cloud customers. The *Cloud Security Alliance* has conducted research on the most important security issues in cloud computing, and released the *Top Threats to Cloud Computing: Egregious Eleven*.[3] Among them, abuse and nefarious use of cloud services are prominent. Hackers are increasingly using legitimate cloud services to engage in illegal activities like malicious mining, which brings great harm to cloud service providers and cloud customers. For example, in 2019, hackers tried to illegally steal the computing power and services of *Amazon Web Services* (AWS) and *Google Cloud* to conduct large-scale cryptocurrency mining activities. As a result, in a very short period of time, its mining business has become one of the largest data usages of AWS in terms of quantity. At the same time, it also caused a total property loss of about $250,000 for some users who used AWS.[4] In addition, the *Mackeeper* security center also issued a report in 2018, pointing out that there are multiple malicious container images containing mining programs in the current popular container platform *Docker Hub*, and with the cumulative download amount of more than five million times of these containers, hackers can deploy mining programs on the developer's computer to run in the background,

* Corresponding authors.
E-mail addresses: lishudong@gzhu.edu.cn (S. Li), tianzhihong@gzhu.edu.cn (Z. Tian).
1 They contributed equally to this work and should be considered co-first authors.
2 IEEE Fellow.
3 https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven.
4 https://bitcoinist.com/singapore-man-faces-34-years-in-prison-for-illegal-crypto-mining/.

https://doi.org/10.1016/j.simpat.2021.102391

Available online 14 August 2021

occupying a lot of resources for mining activities. The report points out that before *Docker Hub* officially removed these container images, the hackers had mined as many as 544 *Monroe* coins and obtained direct economic benefits equivalent to $90,000.[5]

In fact, cloud services are being subjected to more and more attacks based on malicious mining codes. *Kaspersky* laboratory's detection system reported that more than 1.52 million users were attacked by mining viruses in 2020, accounting for 2.49% of all attacks and 13.82% of high-risk programs.[6] In addition, the *Rising* cloud security system reported that a total of 9.22 million mining virus samples were intercepted in 2020, which increased by 332.32% compared with the same period in 2019.[7]

The rise of malicious mining code can be attributed to the following three reasons. (1) Value-added and anonymity of virtual currency. In recent years, the price of various types of digital cryptocurrencies has skyrocketed. According to the statistics of *Coinmarketcap*,[8] in 2020, the price of bitcoin once exceeded $50,000 per BTC, and its market value reached the highest point in history — $920 billion, which is ten times that at the end of 2019. At the same time, *Monroe* coin, which is recognized as the only digital cryptocurrency that cannot be traced at present, has also increased nearly six times in price in the same period, which makes the intrusion into the network through malicious mining code become more and more sought after by cybercriminals. (2) Operability and low cost of mining technology. At present, most hackers tend to use open-source programs, register a wallet address, and then they can easily perform mining activities and obtain great profit, which makes it almost unnecessary for cybercriminals to invest too much energy into it. (3) Emerging in endlessly and easy-exploited system-level vulnerabilities. RCE vulnerability permits hackers to inject operating system commands or malicious codes directly into the background server to control the background system, which has become the most commonly used intrusion channel of mining Trojans. For example, *Z0miner*, a mining Trojan gang, used the unauthorized Command Execution Vulnerability of Weblogic (CVE-2020-14882/14883) to attack, and the time of launching the attack was only 15 days after the official security announcement of Weblogic (2020.10.21). Similarly, the mining Trojan named *Runminer* has attacked about 16,000 cloud servers in 2020 by using Apache Shiro deserialization vulnerability (CVE-2016-4437).[9]

Malicious mining code usually shows the following three characteristics in the cloud platforms: (1) Multiple ways of transmission. Malicious mining code can be spread through vulnerability, weak password blasting, software bundling, social engineering, and many other channels. In addition to mining modules, malicious mining code also carries worm modules with horizontal propagation ability, which can spread wantonly through the vulnerability in the cloud servers and infect more targets. (2) Attempt to hide harmful behaviors of profit-seeking. While malicious mining code is running, cloud customers can easily be aware of it because that the system usually gets stuck as the code takes up a lot of resources of cloud services. Therefore, it will use technologies such as disguising as system files and no file persistence to protect itself. Even if found by users, it will not be easily cleared, so it can occupy available resources for a long time and obtain benefits from mining. (3) Diligent in updating. At present, most of the common malicious mining code will be updated for a long time, such as changing the mining currency at any time, adding more ways of transmission, and adding confusion for self-insurance.

Aiming at various security and privacy issues in the cloud computing environment, academia and industry are exploring effective solutions. For example, according to the information of vulnerability and network environment, an evidence reasoning network can be constructed to track events for detecting lateral movement in the cloud platforms [1]. In addition, cloud computing, fog computing, and network things can be integrated to fulfill verifiable data deletion and flexible access control for sensitive data [2]. However, there is still a lack of effective solutions for the abuse and nefarious use of cloud services for mining activities. It is desirable that recent works tend to use data analysis and machine learning techniques to detect those malicious behaviors feasibly, but the accuracy and robustness of detection also need further sufficient consideration and exploration.

Therefore, in view of the characteristics and trends of malicious mining code in the cloud platforms, a method for detecting malicious mining code is proposed in the paper. The main contributions are as follows:

- We propose a novel and efficient method for detecting malicious mining code, which uses the ensemble learning strategy, has accurate and efficient performance, and can be simply deployed in the cloud platforms to detect the abuse and nefarious use of cloud service.
- We propose an effective method of feature engineering for malicious mining code. The features focus on the characteristics of string distribution obtained through data analysis, making this method simpler and faster.
- We devise a model for detecting malicious mining code, which implements a strategy for enhancing robustness based on Bagging and Boosting algorithms. By randomly extracting samples and letting base models vote together to decide, the variance of model detection can be reduced obviously.
- We conduct a series of experiments including accuracy and robustness evaluation. The experimental results demonstrate that the proposed method outperforms all existing algorithms with a large margin.

The paper continues with Section 2, describing the related works. In Section 3 the methodology for detecting malicious mining code is introduced. Section 4 presents the experimental analysis of the study. Conclusions and future works are drawn in Section 5.

[5] https://mackeeper.com/blog/cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-attackers/.
[6] https://go.kaspersky.com/rs/802-IJN-240/images/KSB_statistics_2020_en.pdf.
[7] http://it.rising.com.cn/dongtai/19747.html.
[8] https://coinmarketcap.com/.
[9] https://s.tencent.com/research/report/1257.html.

2. Related works

2.1. Malicious mining code detection

At present, the working mechanism of malicious mining code mainly includes web mining scripts and executable mining files. For the web mining script, when it is embedded in the web page and visited by users, the browser will parse its content and then execute it, which will cause the browser to occupy a lot of computer resources for mining. For the executable mining file, it can be divided into mining file that launches attacks passively or actively according to the attack types. Passive-attack mining files mean that users cannot start mining activities until running the executable Trojan program. Many of these files are profit-inducing and usually disguise as "urgently needed" applications like game plug-ins and activation tools, or applications that can enable users to obtain benefits directly or indirectly like hang-up software, and Internet cafe VIP video player. While active-attack mining files are contrary, one of the typical representatives is the botnet of mining Trojan. Specifically, hackers invade the computer and implant the mining Trojan, and then use the invaded computer to continue to implant the mining Trojan into other computers, and finally build the botnet.[10]

Based on the working mechanisms mentioned above, a lot of methods for detecting malicious mining code have emerged, which can be divided into traditional detection and machine learning detection. The traditional detection type refers to the detection method based on the signature, which aims to match some key features of the malicious mining code to the known ones. For example, Qin et al. [3] proposed a detection method for web mining scripts, which used *Python* regular expression matching to identify mining *JS* scripts of web background code. While the detection type based on machine learning is usually applied for the generalized malicious code (including but not limited to malicious mining code) detection, including methods of static and dynamic analysis. Static analysis is to read the contents of the program file directly without executing the program and determine whether the file has a malicious function by analyzing the instruction and structure of the program. Dynamic analysis is to determine whether the file has a malicious function by analyzing the running behavior of the program file in the isolated environment (such as simulator and sandbox) [4]. In terms of static analysis, Shabtai et al. [5] disassembled the PE file to obtain the n-gram statistical characteristics of the file opcode to help detect unknown malware. Zhang et al. [6] proposed a method for malware detection based on feature fusion by using CNN and BPNN to embed the features of opcode and API call, which could effectively detect malware and its variants. Martinelli et al. [7] proposed a ML-based method to identify mobile malware, which extracted a set of features counting the occurrences of a specific group of opcodes extracted from the code of Android apps. Li et al. [8] proposed a method applying unsupervised learning to perform detection tasks, which was very enlightening to the tasks of malware detection and family clustering. In terms of dynamic analysis, Ravi et al. [9] proposed a dynamic malware detection system, which could obtain Windows API call sequence through real-time monitoring and generate classification rules by association mining algorithm for the malware classification task. Uppal et al. [10] also carried out feature extraction and feature selection for API call sequences and constructed a malicious code detection model based on many machine learning techniques. Duan et al. [11] implemented a DLL-based framework through the Hidden Naive Bayes algorithm, which could classify benign and malicious processes automatically and then explain malware behaviors at a higher semantic level. Shafiq et al. [12–15] paid more attention to the traffic generated from malware and proposed several methods based on machine learning for malicious traffic identification. For malicious mining code, Gao et al. [16] tried to fuse the static features of web mining script and dynamic features of mining behavior and then trained a model to detect malicious mining code by comparing several algorithms for classification.

From the existing methods for detecting malicious mining code, the traditional detection type performs fast and has a low false-positive rate, but it needs to rely on a large number of existing expert experience, so it can only detect the known malicious mining code. Recently, malicious code gradually increases and performs much more complex forms, leading to machine learning techniques having more advantages in the field of malicious code detection than traditional ones. To begin with, the static analysis method does not need to execute files, which has a fast detection speed and does not produce malicious behavior that probably endangers the system. General malicious codes usually call specific system functions to achieve malicious behavior, so it is easier to locate their API features. However, mining behavior is mainly based on cryptographic operations, and its features lie in the instruction sequences. Relatively speaking, effective feature extraction of mining code has higher requirements for analysts. While the dynamic analysis method can deeply analyze the behavior of malicious code, which could acquire higher accuracy. However, this method must rely on a secure execution environment and need to continuously monitor the dynamic behavior of malicious code, which will cause a lot of computing resources to be wasted and more time-consuming. From the perspective of detection objects, the existing methods are more focused on web mining scripts and lack effective methods for detecting executable mining files.

2.2. Model robustness

When it comes to model robustness, Huber gives three levels of interpretation from the perspective of robust statistics [17]: (1) The model has relatively high accuracy, which is the basic requirement for a machine learning model. (2) The relatively small deviation of the model assumption can only have a delicate impact on its performance. (3) The relatively large deviation of the model assumption cannot have a "catastrophic" impact on its performance. Recently, as artificial intelligence and machine learning become prevailing, the research and evaluation of model robustness have gradually become the focus of public attention. For example, Wang

[10] https://zt.360.cn/1101061855.php?dtid=1101062360&did=610067442.

et al. [18] summarized the current researches on model robustness in AI adversarial environment and gave several mainstream research directions to improve model robustness. Zheng et al. [19] constructed a malware family classification model with higher robustness by randomly inserting API sequences to generate simulation adversarial samples.

However, for most of the machine learning models, on the one hand, it is still a very complex nonlinear optimized process to improve robustness by constructing adversarial samples, and there is no sufficient theoretical basis to prove that the robust improvement achieved by this means is reliable; on the other hand, from the perspective of the model designation, we can improve and optimize the structure of the model for the purpose of enhancing robustness, but related research work has been relatively rare by now.

3. Malicious mining code detection method

This paper introduces a robust method for detecting malicious mining code based on ensemble learning. Firstly, the method reads the original string by static analysis of data, extracts the features, and vectorizes the obtained features by applying the TF–IDF algorithm combined with n-gram. After that, the method constructs a detection model by applying optimized strategies based on a variety of ensemble learning algorithms. Finally, an accurate, stable, and efficient malicious mining code detection model can be obtained.

3.1. Feature engineering

The dataset[11] for research comes from a large number of PE programs containing malicious mining code and non-mining code captured from the Internet, which includes 2,000 malicious mining code samples (black samples) and 4,000 non-mining code samples (white samples). The MZ header, PE header, import and export table, and some other areas in the sample PE structure have been erased, which indicates that dynamic analysis of the samples cannot be carried out. Nevertheless, the code instruction characteristics of its mining function still exist, so it is effective to use the static analysis method to process the samples.

3.1.1. Data preprocessing

The idea of data preprocessing with static analysis is to treat the PE file as a kind of binary file, read the file in the form of binary bytecode, and then decode it into the string to explore and extract features from the string level. On the one hand, for binary files, string refers to the sequence of printable characters in the file. The string of binary files usually contains some key information, such as commands (like HTTP and FTP) to download web resources, IP address and hostname to reveal the address information of program connection, text to explain the purpose of binary files, the compiler that creates binary files, and the programming language, embedded script or HTML used to write binary files. On the other hand, in addition to the common system API calls, malicious mining code usually needs to call some non-system APIs and DLLs, so as to realize the functions such as process creation, memory allocation, registry modification, and the printable string also reflects some mining behavior characteristics to a certain extent. In 2020, *Tencent Cloud Workload Protection* captured a new mining Trojan *Loggerminer* that spreads and launches attacks in the cloud host. *Loggerminer* features exactly that the *looger* string is widely used in the code as the system account name, file path, communication domain name, etc.[12] Therefore, it is feasible to explore the string information in the executable mining file to acquire effective features, so as to quickly understand the possible situation.

3.1.2. Exploratory data analysis

Based on the above ideas, we can make statistical analysis on the string features to explore whether there are some significant characteristics in the string distribution of malicious mining code and other codes. First of all, through the coarse-grained statistical analysis, we can find that there are some differences between black samples and white samples in the type and quantity of strings. The analysis results are shown in Figs. 1 and 2.

Furthermore, through fine-grained statistical analysis, we can find that there is a significant difference in the distribution of strings between black samples and white samples. As shown in Figs. 3 and 4, malicious mining code contains more strings such as *allocation* and *error*, while non-mining code contains more strings such as *SUCCESS* and *registry*. The analysis results also verify the original conjecture, so we can take the string distribution information as an effective feature to distinguish malicious mining code from other codes, and regard it as the core rule to construct the detection model.

[11] https://datacon.qianxin.com/opendata/maliciouscode.

[12] https://s.tencent.com/research/report/1177.html.

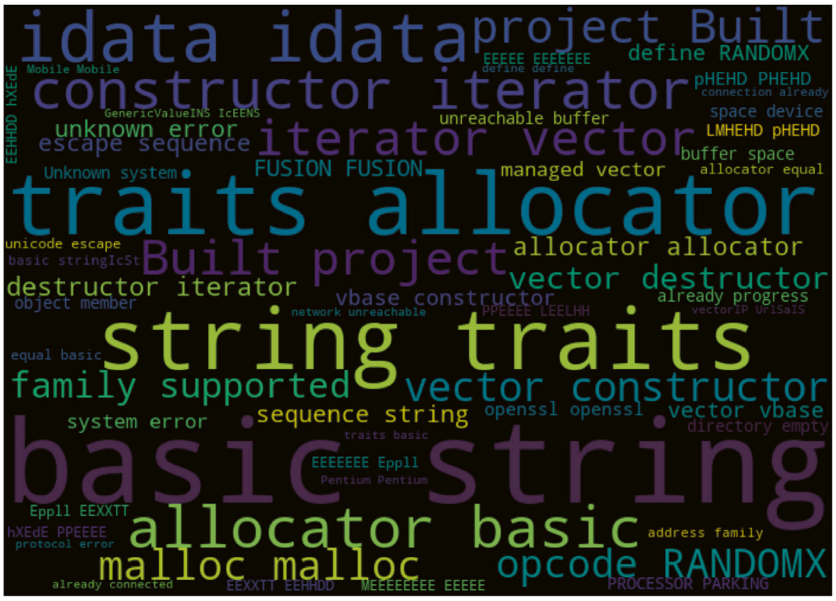


Fig. 1. Distribution of malicious mining code strings.

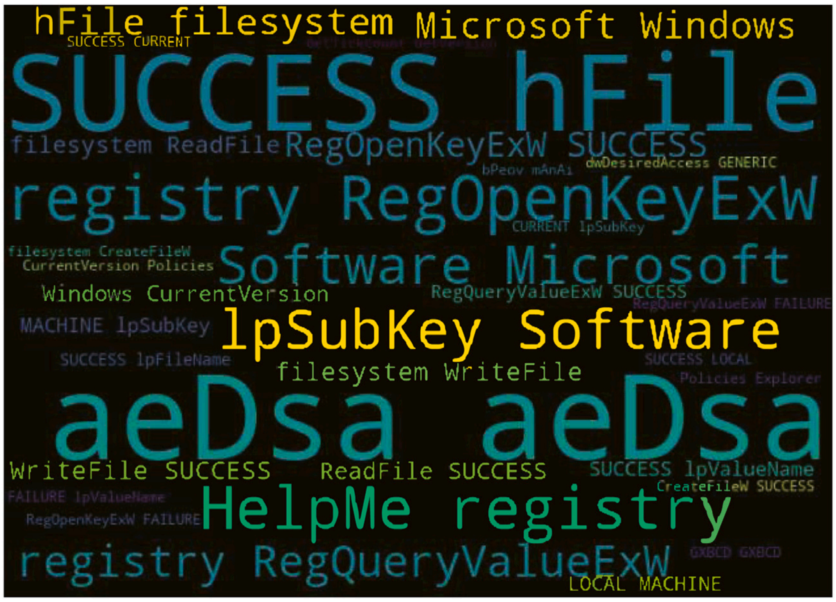


Fig. 2. Distribution of non-mining code strings.

3.1.3. Feature construction

For the string in the sample, this paper selects the n-gram model (a statistical language model) to process it. N-gram is to carry out a sliding window operation on the content of the text according to the size of n bytes to construct a sequence of n-length byte fragments. Each byte segment is called a gram. Firstly, acquire statistics of the occurrence frequency of all grams, and then filter them according to the preset threshold, leading to the gram sequence set (that is, the feature vector space of the current text). The n-gram model derives from the *Markov hypothesis*: the appearance of the nth word is only in connection with the previous n-1 word, regardless of any other word. Suppose that the sentence s in the text consists of word sequences $(w_1, w_2, \dots, w_m)$, and then the probability of the sentence $P(s)$ is the product of the probability of the appearance of each word w_i in the word sequence, and

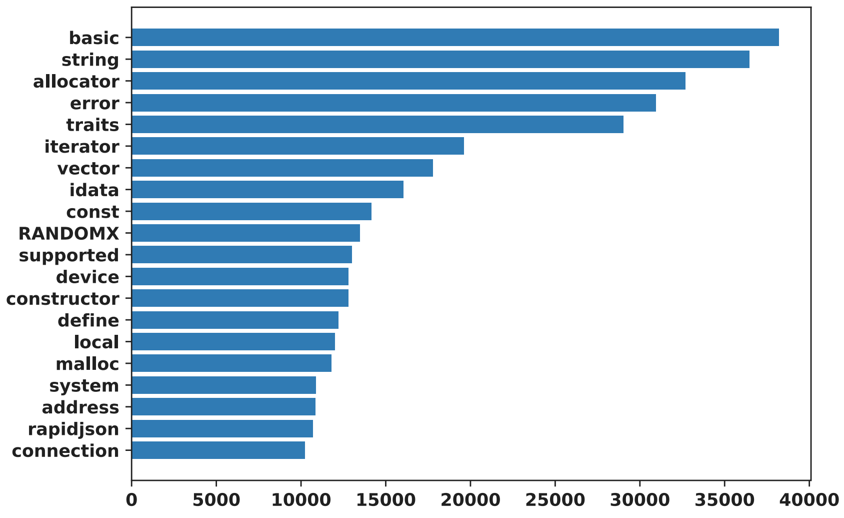


Fig. 3. Malicious mining code string frequency statistics.

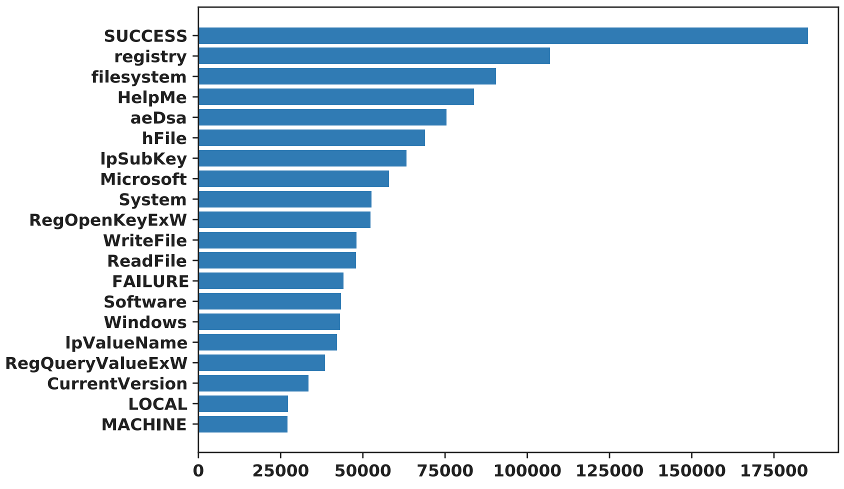


Fig. 4. Non-mining code string frequency statistics.

the latter can be obtained directly from the corpus. Use n-gram to calculate the probability $P(s)$ as Formula (1).

$$\begin{aligned} P(s) &= P(w_1, w_2, \dots, w_m) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_m|w_{m-n+1}, w_{m-n+2}, \dots, w_{m-1}) \\ &= P(w_1)\prod_{i=2}^{m} P(w_i|w_{i-n+1}, w_{i-n+2}, \dots, w_{i-1}) \end{aligned} \tag{1}$$

The common n-gram models are bi-gram and tri-gram, which correspond to different values of n respectively.

After the n-gram process, we get the string distribution feature of the samples, and then we vectorize it through the TF–IDF algorithm. TF–IDF is a statistical algorithm indicating how a word matters in the documents of the corpus. We apply the principle of TF–IDF to the string in the samples. The basic idea is that the importance of the string in the sample increases directly with the times it emerges in the sample but decreases inversely with the times it emerges in all samples.

First, for the original string in the samples, the n-gram sequence can be obtained through the above method, and then the TF and IDF of each string are counted respectively. The TF and IDF are calculated as Formula (2) and Formula (3) respectively.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2}$$

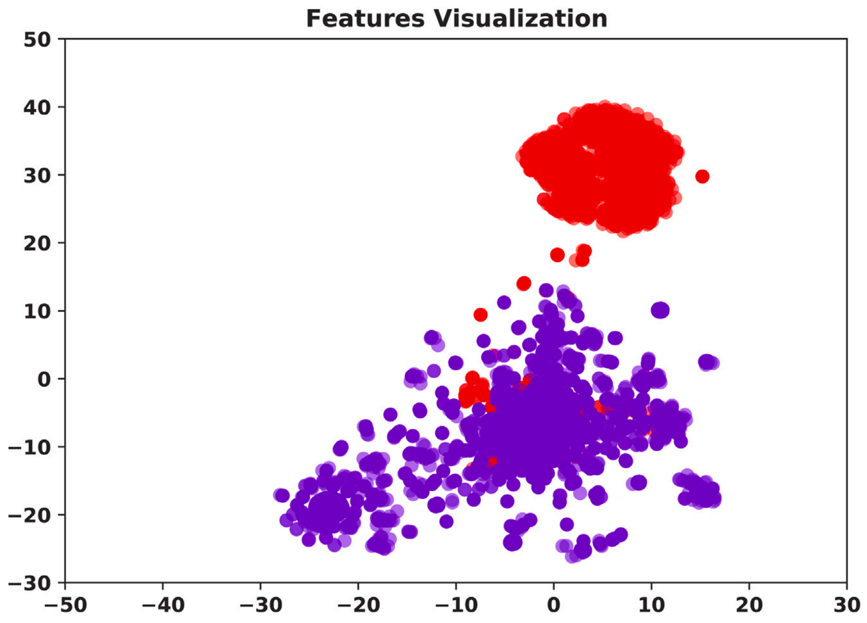


Fig. 5. Feature visualization results.

Where $TF_{i,j}$ represents the frequency of string i in sample j; $n_{i,j}$ is the times that string i emerges in sample j, and $\sum_k n_{k,j}$ means the total amount of strings in sample j.

$$IDF_i = \log \frac{|D|}{1 + |\sum i \in j|} \tag{3}$$

Where $|D|$ is the total amount of samples, and $|\sum i \in j|$ means the number of samples containing string i. In order to prevent the denominator from being 0, we add 1.

The TF–IDF value of each string is equal to multiplying the values of TF and IDF. After calculating the importance of the sample string, we can get the string feature matrix of malicious mining code and non-mining code, which will be input into the detection model.

To verify the effectiveness of the above feature engineering, we attempt to visualize the feature matrix through the t-SNE algorithm. T-SNE algorithm can capture the local structure of high-dimensional data well and reduce high-dimensional data to lower dimensions (like 2D), so as to realizing visualization [20]. The effect of feature visualization is shown in Fig. 5. It can be seen that the above feature engineering nearly makes all the sample data points present two clusters, which indicates that the feature engineering has achieved excellent processing results.

3.2. Model construction

3.2.1. Ensemble learning

Ensemble learning methods train multiple learners and combine them to solve a problem, the typical representatives among which include Bagging and Boosting [21]. Bagging algorithm randomly extracts samples from the training dataset, uses the samples to train the model, and finally all the models vote together to determine the predictive value. Boosting algorithm builds a series of models and aggregates them to get a stronger learner with better performance. In the process of fitting, each model in the sequence will pay more attention to the observation data that the previous model predicted wrong.

The learner based on ensemble learning integrates the advantages of multiple base learners and has stronger generalization ability (that is, higher accuracy and better robustness) and parallel ability. Assuming that the error rates of base learners are mutually independent, based on *Hoeffding's inequality*, that of ensemble learner can be calculated as Formula (4).

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor T/2 \rfloor} \begin{pmatrix} T \\ k \end{pmatrix} (1-\epsilon)^k (\epsilon)^{T-k} \leq exp(-\frac{1}{2}T(1-2\epsilon)^2) \tag{4}$$

Among them, $H(x)$ is the result of voting by the base learners, $f(x)$ is the true label of x, and T is the number of base learners. From the right end of the formula, when the amount of base learners T is enough, the total error rate tends to 0, which shows that under certain conditions, the learner based on ensemble learning can achieve a higher accuracy as far as possible.

The enormous advantages embodied in ensemble learning provide feasible optimized strategies for solving many problems in the network environment. For example, for web attack detection problems, combine all the results of multiple deep models by an ensemble learner to acquire the ultimate decision, and the web attacks can be detected accurately with lower false-positive rates and false-negative rates [22]. Aiming at the problem of cloud computing resource prediction, perform an intelligent and effective algorithm to integrate existing basic models, and we can obtain a higher accuracy of resource prediction in the cloud environment [23]. For the problem of malware classification, the multi-model ensemble method can obtain higher classification accuracy [24].

3.2.2. Bias–variance decomposition

The previous part explained the related concepts of ensemble learning and the theoretical basis for improving the accuracy, but the principle of improving the robustness still needs to be explored. Bias–variance decomposition plays a significant role in explaining the generalization performance of learning algorithms. Bias refers to the difference between the expected output and the true label. It measures the deviation degree between the expected prediction and the real result of the learning algorithm and describes the fitting ability of the learning algorithm itself. Variance refers to the varied range of expected output and can measure the variation of learning performance that is caused by the variation of the training set of the same size, so as to indicate the effect of data disturbance [21]. Bias–variance decomposition attempts to decompose the expected generalization error of the learning algorithm. Assuming that the noise expectation is 0, this generalization error can be represented as the sum of bias and variance. The expression of expected generalization error E is shown in Formula (5).

$$\begin{aligned} E(f;D) &= E[(f(x;D) - y_D)^2] \\ &= E_D[(f(x;D) - \bar{f}(x))^2] + E_D[(\bar{f}(x) - y)^2] \end{aligned} \tag{5}$$

Where $f(x;D)$ represents the prediction of model f on sample x, which is obtained for training set D. y_D is the label of x in the dataset, y is the true label of x, and $\bar{f}(x)$ is the expected prediction of model f.

Bias–variance decomposition shows that the generalization performance depends on the learning ability of algorithms, the sufficiency of data, and the difficulty of learning tasks. Given the task, to obtain better generalization performance, we need to make the bias smaller, so as to fully fit the data and enhance the accuracy of the model prediction. At the same time, we also need to make the variance smaller, so that the impact of data disturbance can be smaller, and the robustness of the model can also be improved.

The idea of bias–variance decomposition can help us better understand the complexity of the model, which is conducive to our exploration and attempt to improve the model. According to bias–variance decomposition, Boosting is considered to focus on reducing bias because its principle is to build a strong ensemble learner based on the learners with weak generalization performance. When it comes to reducing variance, Bagging is preferred cause that its principle is to average the models trained on multiple groups of randomly sampled data.

The principle of reducing variance in Bagging can be analyzed through probability theory. Suppose the mean and variance of random variable X is μ and σ respectively, then for N samples which are independent identically distributed, the sample mean $\bar{X}$ is shown in Formula (6).

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{6}$$

Further, the expectation $E(\bar{X})$ and variance $D(\bar{X})$ can be calculated as Formula (7) and Formula (8) respectively.

$$E(\bar{X}) = \frac{1}{N}\sum_{i=1}^{N} E(X_i) = \mu \tag{7}$$

$$D(\bar{X}) = \frac{1}{N^2}\sum_{i=1}^{N} D(X_i) = \frac{\sigma^2}{N} \tag{8}$$

It can be seen that the expectation of the sample mean is the same as that of the sample, but the variance of the sample mean is only $\frac{1}{N}$ of the variance of the sample.

The principle of Bagging is to average the models trained on multiple groups of randomly sampled data. Therefore, the variance of the model can obviously be reduced. Considering that part of the training samples extracted by Bagging may be the same, each base model will not be completely independent. Assuming that the number of base models is M and the correlation between models is ρ, the sample variance realized by the Bagging strategy is shown in Formula (9).

$$D(X) = \rho \times \sigma^2 + (1-\rho) \times \frac{\sigma^2}{M} \tag{9}$$

3.2.3. Detection model

By now, we learn that using the Boosting algorithm to construct the detection model can achieve higher accuracy by reducing the bias. On this basis, we can take further optimized strategy, that is, using the Bagging algorithm to reduce the variance to strengthen the robustness of the model, which contributes to reaching the bias–variance tradeoff of the model prediction. At this time, the detection model can acquire higher accuracy and better robustness in theory. The proposed model for detecting malicious mining code is shown in Fig. 6.

Specifically, after obtaining the input of the string feature vector, the model firstly uses the Bagging algorithm to randomly extract training samples and then uses Boosting algorithm (like XGBoost) to carry out the actual training task of the model. The trained model is taken as one of the sub-models, and the sub-model is used for this round of prediction. The above process is defined as a decision cycle of the whole model detection process. According to the Bagging strategy, the sample prediction of multiple decision cycles is completed, and we can finally acquire the result of the detection model by averaging multiple decision results.

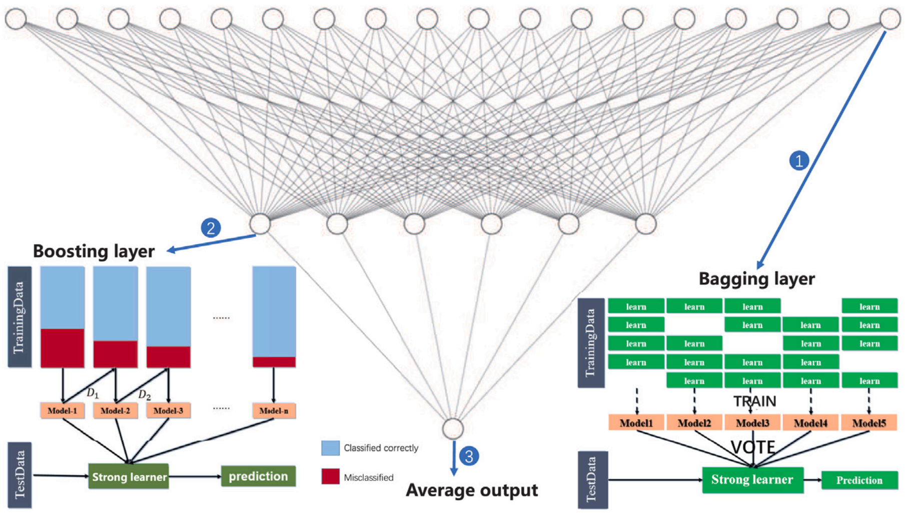


Fig. 6. Malicious mining code detection model.

4. Experimental results and analysis

In this section, two groups of experiments are designed to analyze the performance of traditional learners and the proposed model fusing ensemble learning strategies on malicious mining code detection. The experimental dataset uses the malicious mining code (black sample) and non-mining code (white sample) sample set mentioned in Section 3. The ratio of samples in the training set and test set is 4:1, with the training set containing 1,600 black samples and 3,200 white samples, and the test set containing 400 black samples and 800 white samples, respectively.

4.1. Accuracy evaluation

The first group of experiments mainly evaluated the classification effect of different models, that is, the accuracy of detection. AUC and F1-score were used as evaluation indicators.

AUC means the area under the ROC Curve. The process of AUC calculation is as follows: (1) Sort the samples according to the results of the model prediction, and then predict each sample to be positive according to this order. (2) Calculate False positive rate (FPR) and True Positive Rate (TPR) each time, and then regard them as horizontal and vertical ordinates of ROC curve. (3) Plot the ROC curve and calculate the area under it as AUC. In real tasks, the ROC curve is usually used to study the generalization performance, and the AUC value is a significant indicator for evaluation. TPR and FPR are calculated as Formula (10) and Formula (11) respectively. F1-score equals a harmonic average based on Precision (P) and Recall (R), commonly used to evaluate the accuracy of the classification model. P, R, and F1-score are calculated as Formula (12), Formula (13), and Formula (14) respectively. By getting a higher value of AUC, we hope to train a model with a lower false-positive rate as far as possible. By getting a higher value of F1-score, we hope to train a model with a lower false-negative rate as far as possible.

$$TPR = \frac{TP}{TP + FN} \tag{10}$$

$$FPR = \frac{FP}{TN + FP} \tag{11}$$

$$P = \frac{TP}{TP + FP} \tag{12}$$

$$R = \frac{TP}{TP + FN} \tag{13}$$

$$F1 - score = \frac{2 \times P \times R}{P + R} \tag{14}$$

Among them, TP, FP, TN, and FN are the number of samples that are true positive, false positive, true negative, and false negative respectively.

In the experiment, we explored the classification accuracy of the model fusing Bagging and XGBoost algorithms. At the same time, we discussed the optimization effect after applying the optimized strategy used in this model to the following traditional machine learning classification models: Naive Bayes (NB), Support Vector Machine (SVM), Linear Regression (LR), K-Nearest Neighbor (KNN), Decision Tree (DT) and Artificial Neural Network (ANN). Results of the comparison are shown in Table 1.

From Table 1 we see that the existing classifiers have significantly improved the classification effect after applying the optimized strategy. Among them, the proposed detection model can achieve an AUC value of 0.992 and an F1-score value of 0.987 after fusing Bagging and Boosting algorithms.

Table 1
Comparison of model accuracy improvement.

Model	Indicator			
	AUC (original)	AUC (optimized)	F1-score (original)	F1-score (optimized)
NB	0.948	0.955	0.914	0.918
LR	0.972	0.974	0.972	0.969
SVM	0.979	0.981	0.978	0.976
KNN	0.976	0.979	0.972	0.972
ANN	0.984	0.989	0.979	0.985
DT	0.984	0.991	0.979	0.982
XGBoost	0.986	0.992	0.984	0.987

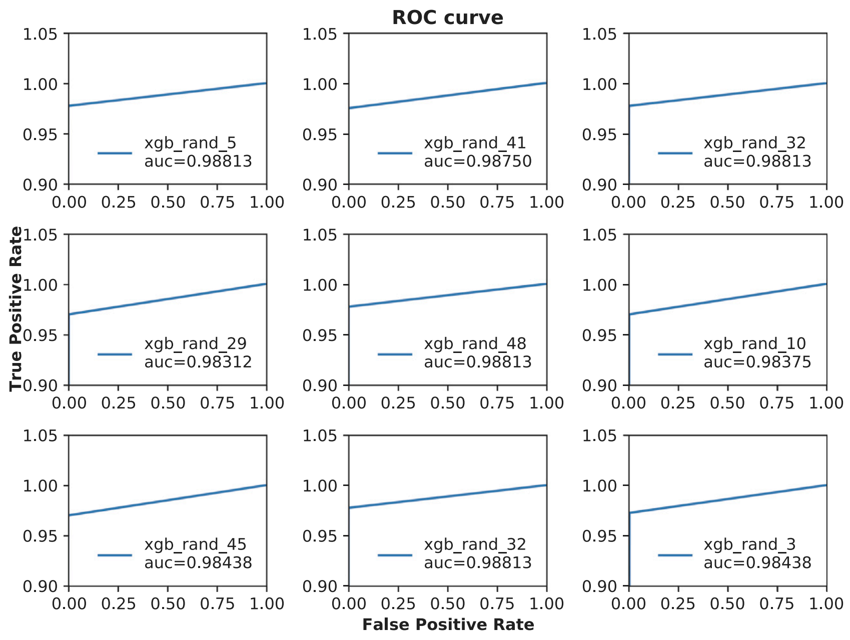


Fig. 7. ROC curve of XGBoost under different random seeds.

4.2. Robustness evaluation

The second group of experiments mainly evaluated the robustness of different models, focusing on the variation of AUC values under different sample data. Here we chose to compare XGBoost with the proposed model. We evaluated the robustness of the model by training models with nine different random seeds, which represented nine rounds of different data input. In Fig. 7, the ROC curves of all nine times for XGBoost are displayed. While the ROC curves of the proposed model under the same set of seeds are shown in Fig. 8. Both models performed the phenomenon of AUC variation to a certain extent. This variation is specifically shown in Fig. 9, with horizontal ordinate representing the order of random seeds and vertical ordinate representing the value of AUC.

The results show that under the same set of seeds, the AUC values obtained by XGBoost have obvious fluctuation in the thousandth, with the range 0.50% and the standard deviation 0.002. While the variation of AUC values is significantly alleviated after fusing Bagging and Boosting algorithms, with the range reaching only 0.26% and the standard deviation declining to 0.0009 of the ten-thousandth order. It means the robustness of the proposed model is improved to a certain extent after applying the optimized strategy.

5. Conclusions and future works

The method for detecting malicious mining code proposed in this study is that firstly, the string in the executable mining file is analyzed and the effective features are extracted, and then the detection model is constructed and trained by fusing Bagging and Boosting ensemble learning algorithms. By randomly extracting samples and letting models vote together to decide, the variance of model detection can be reduced obviously. The experimental results show that this detection model can get higher AUC and F1-score, and the variation of AUC values is more delicate under different data input, which indicates that the proposed method

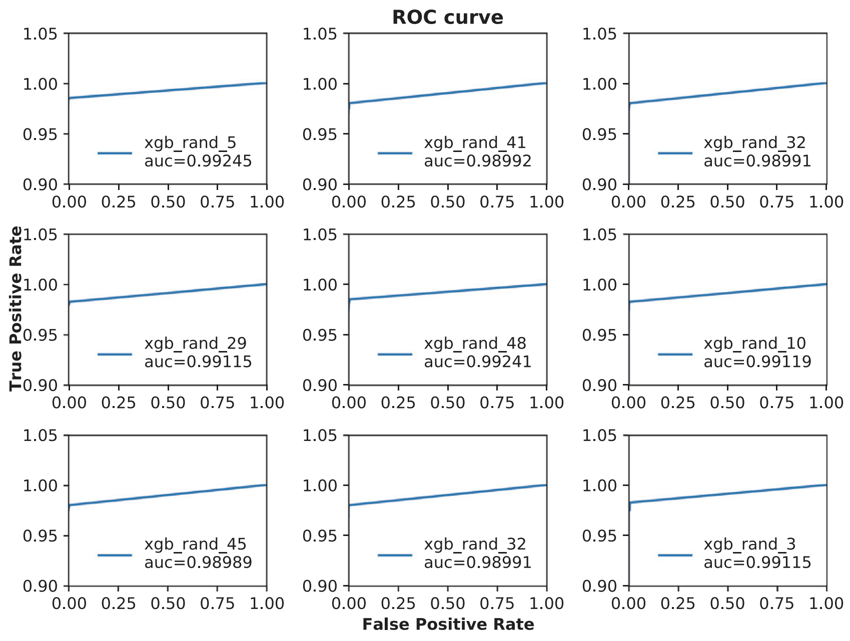


Fig. 8. ROC curve of fused model under different random seeds.

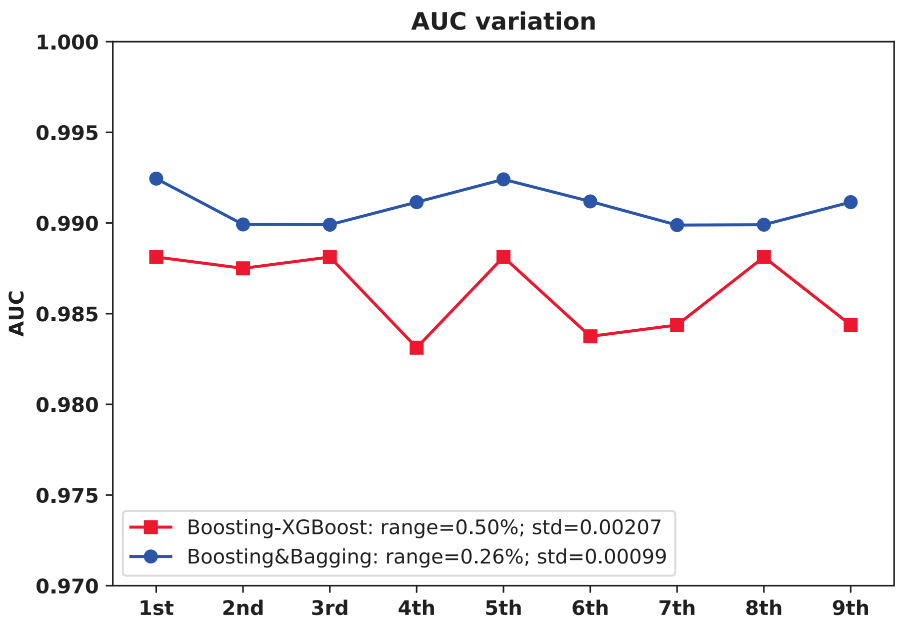


Fig. 9. AUC variation of XGBoost and fused model.

has higher accuracy and better robustness. In addition, from the perspective of static analysis, the feature engineering and model construction process is relatively simple, and the model prediction is also efficient, which indicates that the proposed method applies to the tasks for detecting malicious mining code in the cloud platforms.

In future work, a robust detection method based on ensemble learning, which does not rely on the file structure and working mechanism of specific malicious code, can be applied to the detection tasks of various malicious codes such as ransomware and Android malware [25–27]. Moreover, we also plan to evaluate the effect of this method on improving model robustness in other issues of cybersecurity with limited data scales [28–31].

Acknowledgments

This research is supported by NSFC, China (No. 62072131, U20B2046, 61972106), Key R&D Program of Guangdong Province, China (No. 2019B010136003), National Key Research and Development Program of China (No. 2019QY1406), Science and Technology Projects in Guangzhou, China (No. 202102010442), Guangdong Higher Education Innovation Group, China (No. 2020KCXTD007), Guangzhou Higher Education Innovation Group, China (No. 202032854), and Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme, China (2019).

References

[1] Z. Tian, W. Shi, Y. Wang, C. Zhu, X. Du, S. Su, Y. Sun, N. Guizani, Real-time lateral movement detection based on evidence reasoning network for edge computing environment, IEEE Trans. Ind. Inf. 15 (7) (2019) 4285–4294, http://dx.doi.org/10.1109/TII.2019.2907754.
[2] Y. Yu, L. Xue, Y. Li, X. Du, M. Guizani, B. Yang, Assured data deletion with fine-grained access control for fog-based industrial applications, IEEE Trans. Ind. Inf. 14 (10) (2018) 4538–4547, http://dx.doi.org/10.1109/TII.2018.2841047.
[3] Y. Qin, L. Liu, H. Gao, S. Liu, Detection and prevention of malicious mining behaviors on web pages, Netw. Secur. Technol. Appl 216 (12) (2018) 54–56, http://dx.doi.org/10.3969/j.issn.1009-6833.2018.12.029.
[4] L. Zhang, Y. Cui, J. Liu, et al., Application of machine learning in cyberspace security research, Chinese J. Comput. 41 (9) (2018) 1943–1975, http://dx.doi.org/10.11897/SP.J.1016.2018.01943.
[5] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, Y. Elovici, Detecting unknown malicious code by applying classification techniques on opcode patterns, Secur. Inform. 1 (1) (2012) 1–22, http://dx.doi.org/10.1186/2190-8532-1-1.
[6] J. Zhang, Z. Qin, H. Yin, L. Ou, K. Zhang, A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding, Comput. Secur. 84 (JUL.) (2019) 376–392, http://dx.doi.org/10.1016/j.cose.2019.04.005.
[7] F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, G. Vaglini, Model checking and machine learning techniques for HummingBad mobile malware detection and mitigation, Simul. Model. Pract. Theory 105 (2020) 102169, http://dx.doi.org/10.1016/j.simpat.2020.102169.
[8] S. Li, L. Jiang, X. Wu, W. Han, D. Zhao, Z. Wang, A weighted network community detection algorithm based on deep learning, Appl. Math. Comput. 401 (7) (2021) 126012, http://dx.doi.org/10.1016/j.amc.2021.126012Get.
[9] C. Ravi, R. Manoharan, Malware detection using windows api sequence and machine learning, Int. J. Comput. Appl. 43 (17) (2012) 12–16, http://dx.doi.org/10.5120/6194-8715.
[10] D. Uppal, R. Sinha, V. Mehra, V. Jain, Malware detection and classification based on extraction of api sequences, in: 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, pp. 2337–2342, http://dx.doi.org/10.1109/ICACCI.2014.6968547.
[11] Y. Duan, X. Fu, B. Luo, Z. Wang, J. Shi, X. Du, Detective: Automatically identify and analyze malware processes in forensic scenarios via DLLs, in: 2015 IEEE International Conference on Communications (ICC), 2015, pp. 5691–5696, http://dx.doi.org/10.1109/ICC.2015.7249229.
[12] M. Shafiq, Z. Tian, A.K. Bashir, X. Du, M. Guizani, Corrauc: a malicious bot-iot traffic detection method in iot network using machine learning techniques, IEEE Internet Things J. 8 (5) (2021) 3242–3254, http://dx.doi.org/10.1109/JIOT.2020.3002255.
[13] M. Shafiq, Z. Tian, A.K. Bashir, X. Du, M. Guizani, IoT Malicious traffic identification using wrapper-based feature selection mechanisms, Comput. Secur. 94 (2020) 101863, http://dx.doi.org/10.1016/j.cose.2020.101863.
[14] G. Sun, Y. Li, H. Yu, A.V. Vasilakos, X. Du, M. Guizani, Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks, Future Gener. Comput. Syst. 91 (2019) 347–360, http://dx.doi.org/10.1016/j.future.2018.09.037.
[15] M. Shafiq, Z. Tian, Y. Sun, X. Du, M. Guizani, Selection of effective machine learning algorithm and bot-IoT attacks traffic identification for internet of things in smart city, Future Gener. Comput. Syst. 107 (2020) 433–442, http://dx.doi.org/10.1016/j.future.2020.02.017.
[16] J. Gao, Y. Sun, R. Wang, D. Yuan, Research on mining detection model of browser based on machine learning, Comput. Eng. Appl. (2020) URL: http://kns.cnki.net/kcms/detail/11.2127.TP.20201223.1331.014.html.
[17] P.J. Huber, Robust Statistics, 523, John Wiley & Sons, 2004.
[18] K. Wang, P. Yi, A survey on model robustness under adversarial example, J. Cyber Secur 5 (2) (2020) 13–22, http://dx.doi.org/10.19363/J.cnki.cn10-1380/tn.2020.05.02.
[19] R. Zheng, Q. Wang, J. Fu, Z. Jiang, R. Su, S. Wang, A novel malware classification model based on deep learning, J. Cyber Secur 5 (1) (2019) 1–9, http://dx.doi.org/10.19363/J.cnki.cn10-1380/tn.2020.01.01.
[20] L. Van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (86) (2008) 2579–2605, URL: http://jmlr.org/papers/v9/vandermaaten08a.html.
[21] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, CRC press, 2012.
[22] C. Luo, Z. Tan, G. Min, J. Gan, W. Shi, Z. Tian, A novel web attack detection system for internet of things via ensemble classification, IEEE Trans. Ind. Inf. 17 (8) (2021) 5810–5818, http://dx.doi.org/10.1109/TII.2020.3038761.
[23] S. Wang, F. Zhu, Y. Yao, W. Tang, Y. Xiao, S. Xiong, A computing resources prediction approach based on ensemble learning for complex system simulation in cloud environment, Simul. Model. Pract. Theory 107 (2021) 102202, http://dx.doi.org/10.1016/j.simpat.2020.102202.
[24] H. Yang, S. Li, X. Wu, H. Lu, W. Han, A novel solutions for malicious code detection and family clustering based on machine learning, IEEE Access 7 (2019) 148853–148860, http://dx.doi.org/10.1109/ACCESS.2019.2946482.
[25] Z. Zhu, F. Zeng, G. Qi, Y. Li, H. Jie, N. Mazur, Power system structure optimization based on reinforcement learning and sparse constraints under DoS attacks in cloud environments, Simul. Model. Pract. Theory 110 (2021) 102272, http://dx.doi.org/10.1016/j.simpat.2021.102272.
[26] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, A. Razaque, Deep recurrent neural network for IoT intrusion detection system, Simul. Model. Pract. Theory 101 (2020) 102031, http://dx.doi.org/10.1016/j.simpat.2019.102031.
[27] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, K. Ren, Android HIV: A study of repackaging malware for evading machine-learning detection, IEEE Trans. Inf. Forensics Secur. 15 (2019) 987–1001, http://dx.doi.org/10.1109/TIFS.2019.2932228.
[28] G. Sun, Y. Zhang, D. Liao, H. Yu, X. Du, M. Guizani, Bus trajectory-based street-centric routing for message delivery in urban vehicular ad hoc networks, IEEE Trans. Veh. Technol. 67 (8) (2018) 7550–7563, http://dx.doi.org/10.1109/TVT.2018.2828651.
[29] D. Wang, B. Song, D. Chen, X. Du, Intelligent cognitive radio in 5G: AI-based hierarchical cognitive cellular networks, IEEE Wirel. Commun. 26 (3) (2019) 54–61, http://dx.doi.org/10.1109/MWC.2019.1800353.
[30] G. Lin, S. Wen, Q.-L. Han, J. Zhang, Y. Xiang, Software vulnerability detection using deep neural networks: A survey, Proc. IEEE 108 (10) (2020) 1825–1848, http://dx.doi.org/10.1109/JPROC.2020.2993293.
[31] S. Li, D. Zhao, X. Wu, Z. Tian, A. Li, Z. Wang, Functional immunization of networks based on message passing, Appl. Math. Comput. 366 (2020) 124728, http://dx.doi.org/10.1016/j.amc.2019.124728.