Hindawi
Security and Communication Networks
Volume 2022, Article ID 9453797, 8 pages
https://doi.org/10.1155/2022/9453797

Research Article

Cryptocurrency Mining Malware Detection Based on Behavior Pattern and Graph Neural Network

Rui Zheng,[1,2] **Qiuyun Wang**,[2] **Jia He**,[1] **Jianming Fu**,[1] **Guga Suri**,[1] **and Zhengwei Jiang**[2,3]

[1]*Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China*
[2]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[3]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Jianming Fu; jmfu@whu.edu.cn

Received 17 September 2021; Revised 12 November 2021; Accepted 9 March 2022; Published 26 March 2022

Academic Editor: Chunhua Su

Miner malware has been steadily increasing in recent years as the value of cryptocurrency rises, which poses a considerable threat to users' device security. Miner malware has obvious behavior patterns in order to participate in blockchain computing. However, most miner malware detection methods use raw bytes feature and sequential opcode as detection features. It is difficult for these methods to obtain better detection results due to not modeling robust features. In this paper, a miner malware identification method based on graph classification network is designed by analyzing the features of function call graph and control flow graph of miner malware, called MBGINet. MBGINet can model the behavior graph relationship of miner malware by extracting the connection features of critical nodes in the behavior graph. Finally, MBGINet transforms these node features into the feature vectors of the graph for miner malware identification. In the test experiments, datasets with different volumes are used for simulating real-world scenarios. The experimental results show that the MBGINet method achieves a leading and stable performance compared to the dedicated opcode detection method and obtains an accuracy improvement of 3.08% on the simulated in-the-wild dataset. Meanwhile, MBGINet gains an advantage over the general malware detection method Malconv. These experimental results demonstrate the superiority of the MBGINet method, which has excellent characteristics in adapting to realistic scenarios.

1. Introduction

Miner malware, also known as cryptocurrency mining hijacking attacks, performs cryptocurrency mining by stealing computing resources from the victim's computing devices [1]. Miner malware has continued to be active in recent years due to the rising value of cryptocurrencies [2]. The continued activity of miner malware poses a significant security threat to user devices. Miner malware stealing computing resources will consume a large number of electric power resources and reduce the lifetime of the victim device [3]. In addition to that, some miner malware can seriously occupy CPU resources and cause system crash to bring substantial business losses to users [4]. Therefore, the detection of miner malware in the wild becomes an important malware detection issue.

Different from other common malware, miner malware has its distinct characteristics. First of all, miner malware has noticeable blockchain computing features compared to common malware, and it needs to constantly interact with the blockchain during the running of the malware. These characteristics will expose a large number of behavior features of miner malware. In local, cryptocurrency computing requires a large number of computing resources and access to the victim's computing devices. Network interaction and local computing features together composed the cryptocurrency mining behavior features of miner malware. In addition, miner malware has unique ecological characteristics, such as reliance on XMR's open-source software to mine Monero cryptocurrencies, which bring distinct characteristics to the behavior pattern of miner malware.

Traditional malware detection techniques mainly rely on signature matching techniques and heuristic detection, but this type of rule matching has the low capability for generalization and finding unknown malware [5]. Machine learning techniques have evolved in recent years and have become another important way to solve malware detection. Furthermore, deep learning has a clear advantage in the large space search problem [6], being able to accurately predict labels in high-dimensional spaces [7]. However, in the malware detection domain, feature selection is still related to the upper limit of the capability of the final training model. Some malware detection studies used a data-driven type of feature to design malware detectors, such as byte-level features [8] and PE format features [9]. Such a feature design idea runs counter to the idea of the traditional malware analysis community, which often determines the class of malware based on its behavior pattern. In addition, some malware research heavily relies on research methods from the traditional field of machine learning, using random splitting and small datasets as test datasets production standards. Such testing methods do not fit the real-world malware detection circumstance, where malware detectors are trained in small datasets and deployed against the scenario of massive malware samples.

In this paper, we use function call graphs and basic block jump graphs within miner malware binaries as features for miner malware detection. The principle of malware detector based on behavior features follows the malware analysis methods of the malware analysis community, while the behavior features are also more difficult to fabricate. In order to explore the role of the behavior features for miner malware detection, we convert the call relationship and instructions jump graph as the feature vector. This approach can highlight the nodes with distinct connectivity properties in the behavior graph. For the evaluation, the large dataset is used as validation samples, and small dataset samples are used as training samples, which follows the paradigm of malware detection. Finally, the popular detection methods for the miner malware and general malware are taken as baselines to represent the effectiveness of the proposed methods.

The contributions of this paper are as follows:

(1) This paper explores the usefulness of behavior pattern on function level and basic block level for miner malware classification under the static analysis. By constructing different graph isomorphic networks to build miner malware classifiers, experiments show that the proposed method in this paper outperforms other baseline methods.

(2) Representative miner malware detection methods and a general malware detection method are chosen for the experiments as baseline methods. The experimental results show that the proposed method achieves advantages over these methods, proving its superiority.

(3) The experiment uses a simulated in-the-wild dataset as the test environment. The experimental results better reflect the effectiveness of the proposed method in the real world.

This paper systematically described a graph-based method for miner malware detection. Section 2 reviews the research on machine learning-based methods for general and miner malware detection. Section 3 introduces the framework of MBGINet, and Section 4 shows the performance evaluation results of all methods on laboratory dataset and simulated in-the-wild dataset. Section 5 summarizes the work of this paper.

2. Related Work

This section first discusses popular malware detection methods based on static analysis features and machine learning. Then, the research of miner malware is described, which contains miner malware ecological development and their detection methods.

2.1. Malware Detection. Static analysis features are important features for malware detection. Compared with dynamic analysis features, static analysis features have the characteristics of fast extraction and variety. Taking the parsing of the PE file as the criterion, we can divide the static features of PE files into three layers, namely, raw bytes, PE structure, and disassembly code. The complexity of parsing rules used for these three feature layers goes from shallow to deep and parsing time from short to long.

Raw bytes are the most accessible malware feature, which does not require format parsing. Raw bytes can be directly taken as the basis for malware identification, but raw bytes often face the challenge of enormous feature dimensions. Therefore, many dimensionality reduction methods are applied to the feature preprocessing of raw bytes, for example, the frequency method [10], information compression method [11], sampling method [12], etc. With the help of these methods, raw bytes are processed into feature vectors suitable for machine learning models to train malware classifiers. Furthermore, some deep learning models [13, 14] can even directly process the original high-dimensional byte features. However, raw bytes have a notorious drawback in that no clear semantic information can be obtained internally. The lack of interpretable malware determination results makes the robustness of this class of methods questionable.

PE structure information is another feature for malware detection. PE structure information needs to be parsed in a fixed format with byte codes to extract feature information compared to raw bytes. The PE header [15] and sections information [9] are useful for malware detection. The extracted PE meta information and sections information describe the basic parsing information of the malware, which has similarities in different malware. However, this information is easy to fabricate, leading to malware detector failure.

The assembly code needs to be further parsed based on the PE structure parsing. By scanning the code sections of the PE file, the disassembly tool can convert binary code to assembly code, even C language pseudocode. The assembly code obtained in this process can represent the behavior of

the malware, including system-level action features and instruction-level action features involved in the malware execution process. From the perspective of program analysis, these features directly determine the maliciousness determination of malware. Moreover, these features are much less likely to be fabricated. Therefore, disassembly code features are one of the desirable types of features for malware detection.

An important feature extracted through disassembly tools is the opcode, which characterizes the sequence of instructions executed by the CPU when the malware is running. Thus, opcode could be taken as the feature to detect malware. Jeon et al. [16] apply a convolutional autoencoder to extract and compress the opcode feature vector. Then, these feature vectors are fed into a recurrent neural network to train a malware detector. However, although the opcode feature can be helpful for malware classification, its inability to describe invocation relationships between instructions limits its usefulness for malware detection.

Among the features obtained from disassembly, graph features constructed by functions or basic blocks can represent the behavior pattern of malware, and such features are used as the basis for malware classification. Yan et al. [17] extract control flow graph from disassembly analysis as classification features and converted code blocks to feature vector using its instructions statistical features. These features are fed into the graph convolutional network to train a malware detector. In addition to the above methods, graph matching algorithms are also used to identify malware. Ammar et al. [18] extract static function call graphs of malware as feature vectors for malware recognition. Graph matching algorithms are used to calculate the similarity between the suspicious samples and the known samples. The similarity is used to determine whether the suspicious sample is malware.

In the tasks of malware detection based on graph feature extracted by disassembly tool, the connections between elements represent that malware's actions can describe the behavior patterns of malware. Such features are robust compared to the raw bytes feature and PE structure feature. Moreover, behavior pattern recognition is an important way to solve malware detection problems.

2.2. Miner Malware Detection. Due to the great danger of miner malware, many studies have been conducted to analyze miner malware activities. Within this process, many features of miner malware have been revealed. For example, Pastrana et al. [19] study the campaigns of miner malware in a decade. The authors find that the cryptocurrency mined by miner malware is concentrated in several currencies such as Monero and Bitcoin. It is not the only case. Tekiner et al. [3] find miner malware focused on XMR or Bitcoin to mine. In addition, miner malware also needs to perform a large number of cryptographic operations when performing tasks. Some standard anti-antivirus measures are also often used before cryptographic operations [20]. These behaviors indicate that the miner malware has a clear pattern of behavior.

For the detection of miner malware, the feature of network traffic [21] and CPU usage [22] have been taken as indictors for detecting miner malware. In terms of static analysis, the opcode is used as an important feature for miner malware detection. Yazdinejad et al. [23] trained a malware classifier using biLSTM for modeling the sequence characteristics of opcode and achieved good experimental results. Naseem et al. [24] introduced a detection method for browser cryptocurrency mining attacks by converting bytecode files into grayscale images and then training CNN models to identify grayscale image features.

Although opcode and grayscale image features can be used as feature vectors for miner malware identification, it is difficult for these features to represent the pattern of miner malware behavior. Since miner malware has obvious behavior characteristics, this paper designs a graph network classification method called MBGINet based on behavior patterns to detect miner malware.

3. MBGINet Architecture

In this paper, we design a Miner Behavior Graph Isomorphism Network (MBGINet) for miner detection. Compared with traditional opcode feature methods and data-driven methods, this approach enables more efficient detection of miner malware and has more robust performance in the real world. In this section, the design of MBGINet will be described in detail. Firstly, the miner malware behavior pattern at the basic block and function level will be described. Secondly, the MBGINet structure and parameters corresponding to the two levels will be presented.

3.1. Miner Behavior Graph Modeling. In the static analysis setting, the behaviors of malware cannot be executed precisely in chronological order. Although there exists a sequence of instructions execution, it is difficult to describe a perfect sequential action due to branching judgment and other issues. Therefore, a graph network becomes the most suitable model to describe the invocation relationship at levels of functions and basic blocks. From the perspective of the malware analysis community, the calling relationships between disassembly codes are the essential source for malware analysis. In the operation of miner malware, the ultimate purpose of all actions performed is to load the cryptocurrency computing module smoothly. Before calling the cryptocurrency mining module, a series of OS information collection and persistence operations need to be performed. This process does not contain much files operation and data exfiltration, which is different from other types of malware. Therefore, miner malware has obvious behavioral differences compared to ordinary software.

The behavior semantic of miner malware can be represented by functions and instructions. The former is the functions call, which is the concrete execution module for behaviors in Ring 3. The latter is the instructions sequence in the assembly code obtained by parsing the binary code,

which is the CPU-level actions. In these two behavior levels, miner malware implements the corresponding malicious capability through function execution and code block execution, respectively.

Function objects are responsible for the execution of specific behaviors, and the boundary of the function is almost the boundary of the behavior. The invocations between functions form the basis of the binary's operation. So the calling relationship between functions represents the entire malware behavior implementation. The calling relationship of functions is clear in miner malware, where not only does the miner malware need to perform the cryptocurrency mining behavior, but some discovery and defense evasion behaviors are also critical to the implementation of the malicious behavior. The combination of these behaviors or the alone blockchain computing brings a distinct behavioral signature representation for cryptocurrency mining malware.

The basic code blocks are a group of instructions with jump instructions as the dividing point. Within the basic code block, instructions are executed sequentially. Among the blocks of instructions, they are executed according to jump conditions. Thus, the instruction-level execution behavior reflects the action sequence of the CPU. Furthermore, the code block generally represents a small execution operation, and the jump operation is related to the conditional execution of the software. This execution paradigm describes the basic malware CPU behavior pattern. CPU behavior pattern represents the behavior of cryptocurrency mining and other malicious operators. CPU behavior patterns are suitable for discovering miner malware, as most miner malware relies on the arithmetic power provided by the CPU to perform blockchain computing [3].

We define the graph classification task model for miner malware identification based on the graph features. Given a graph $G = (V, E)$, node $v_i \in V$, v_i represents a function body or a basic code block, and $E = (v_i, v_j)$ represents the direct call or jump relationship of the nodes. Such a graph network consisting of nodes and edges can represent the behavior pattern of a malware binary. Connectivity relations are used to describe the graph isomorphism between miner malware samples, where all nodes are taken as the same vector. We design MBGINet for modeling the behavior pattern of the control flow relationship in miner malware.

3.2. MBGINet Model Structure. In order to extract the connection relationship between nodes, we choose graph isomorphism network [25] as the classification network. Inspired by the design of the GIN model, we construct graph feature aggregation to collect aggregate features to important nodes and convert the nodes vector to graph embedding for classification. Node feature aggregation layer is shown in equation (1). Each layer further aggregates the node's degree information to the critical nodes and finally converges the structural information of the graph into a vector. In equation (1), k denotes the index of layers and ε denotes the merge coefficient between neighborhood nodes and center node, which is assigned a fixed scalar. h_v^k denotes the message passing operator of graph features. Message passing of graph data in MLP guarantees that the entire layer can be used as an infective function to extract aggregated features of nodes.

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1 + \varepsilon^{(k)}\right) * h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}\right). \quad (1)$$

In the behavior graph, the most remarkable information is the connectivity of nodes, which are aggregated into a vector to describe the graph structure in the task of malware classification. In the further compression of the nodes feature, we extract the maximum value as an input (as shown in equation (2)) to the next fully connected layer, unlike the original GIN model. The operation of global max-pooling is partly due to the max nodes features, leading to much differentiation between miner malware and others. Moreover, the global max-pooling enables highlighting the information describing the calls or jumps in the behavior pattern of miner malware. Behavior graphs at the basic block level have more nodes and more edges than that at the function level; this implies a more sparse connection pattern at the function level. Therefore, we use global max-pooling for graph isomorphism network at the basic block level and use summation pooling for graph isomorphism network at the function level.

$$\text{inter\_layer} = \text{Global Max Pooling}\left(h_v^{(k)}\right). \quad (2)$$

Inspired by the GIN model and after parameter tuning, we designed the framework structure of MBGINet. We separately designed different graph neural network structures for the function level and the code basic block level because of the huge difference between their node sizes. As shown in Figure 1, the structure and hyperparameter of MBGINet-FCG are present. MBGINet-FCG uses a hidden layer of nodes aggregator layer and the nodes are extended to 8-dimensional channels in the MLP layer. As mentioned earlier, the network structure is designed to use sum pooling to handle the convergence features due to the sparse connectivity of the function layer.

Aiming to process the structure of the basic block nodes, four hidden layers with Multilayer Perceptron (MLP) are used to extract aggregated features of the nodes. Global max-pooling is used to extract the largest features that are output as graph classification results through the fully connected layer and activation function. The global max-pooling facilitates the extraction of connectivity features between basic block nodes, and thus the feature vector of the entire invoked graph features is fed to the fully connected layer for classification. As shown in Figure 2, MBGINet-CFG contains four hidden layers with aggregation computation and a global max-pooling layer, where k represents the index of the layer and d represents the number of channels of the MLP in the hidden layer. As depicted in Figure 2, the four hidden layers of the model use a recursive structure to output the final results for the global max-pooling layer instead of using the concatenating output of every hidden layer in the original paper. As a result, this allows for better extraction of

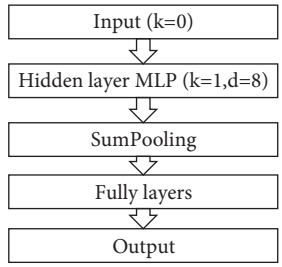

Figure 1: The structure of MBGINet at level of function call graph.

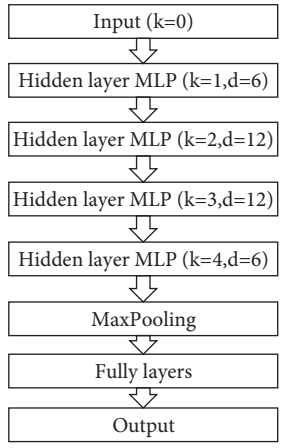

Figure 2: The structure of MBGINet at level of control flow graph.

the most significant node connectivity features through multilayers message passing.

We used the MBGINet structure to model two-level graph pattern of miner malware. In the next section, the performance of these two graph neural networks with different feature levels is evaluated in the same experiments framework.

4. Evaluation

To verify the performance of MBGINet in behavior levels of FCG and CFG, we conducted experiments on two datasets, which simulated the developments environment of the miner malware detector in the real world. Through comparing the performance of MBGINet with baseline methods, we found the MBGINet outperforms opcode sequence model [23], grayscale image feature model [24], and the excellent Malconv model [8]. In this section, we first introduce the datasets used for the experiments and the evaluation metrics of the results. Secondly, the performance of the MBGINet and the baseline methods on the laboratory dataset and simulate-wild dataset are present, respectively. Finally, the time consumption of all methods involved in this paper is described.

4.1. Dataset and Experiments Setup. In order to understand the effectiveness of MBGINet from the perspective of practical applications, this paper uses two miner malware datasets with different data volumes as the benchmark. This is different from some malware detection works [26, 27] that randomly split the training and test samples from one dataset, and the test dataset is always undersize. These two miner malware datasets are published on the malicious code analysis direction of the Big Data Security Analysis Competition ∗ . We call the smaller dataset lab dataset, and call the larger one the simulated in-the-wild (Sim-Wild) dataset.

The number distribution of samples in the lab dataset and the simulated in-the-wild dataset is shown in Table 1. The two datasets consist of miner and not-miner samples, where not-miner one includes other types of malware and benign software. The lab dataset and the simulated in-the-wild dataset have almost the same data distribution ratio; the ratio of miner samples to not-miner samples is about 1 : 2. At the same time, the simulated in-the-wild dataset has about three times the volume of the lab dataset, i.e., 6000 vs. 17,657. Such a data distribution is consistent with developing and deploying machine learning malware detectors in the real world. These malware detectors are always developed on a small dataset and deployed against the massive malware samples in the wild. In this paper, opcode, FCG, and CFG features are extracted using IDA pro †. As shown in Table 1, a small number of samples suffer parsing failure in the feature extraction.

Typical performance metrics are used as evaluation indicators for experiments, such as accuracy, precision, recall, F1-score, and AUC. Since these metrics are usual information retrieval metrics, we do not describe the formulas of these metrics in detail. For the evaluation of time consumption, we compare by calculating the sum of feature extraction, training time, and inference time.

4.2. Performance Comparison on Lab Dataset. In the development of machine learning malware detectors, small datasets are often used for cross-validation experiments. As described previously, we use the lab dataset as a small training and testing dataset to simulate the development of miner malware detectors in a real scenario. The lab dataset is divided into five pieces for performing cross-validation experiments. Four pieces of the dataset are treated as a training dataset, and the remaining one piece is used to test trained model in each experiment. The average of the results of the five experiments is taken as the performance representation. The results of various methods are shown in Table 2.

Table 2 shows the detection results of five machine learning methods, where MBGINet-FCG and MBGINet-CFG denote the effects of MBGINet on two levels of graph features, and the remaining three models are baseline methods. The grayscale image (GI) method is derived from [24], which detects cryptocurrency mining attacks in browsers by converting bytecode files to grayscale images to perform the classification task. The OP denotes a dedicated miner malware detection method [23] that implements classifier construction by using opcode as a feature vector using a bidirectional LSTM method. Malconv [8] is a representative approach for general malware detection. In this experiment, Malconv takes raw bytes of binary files as

Table 1: Number of samples in the lab dataset and the Sim-Wild dataset for different features.

Experiments	Lab dataset		Sim-Wild dataset	
	Miner	Not miner	Miner	Not miner
Original	2000	4000	5898	11759
FCG	1885	3773	5167	11120
CFG	1885	3773	5167	11120
Opcode	1885	3699	5823	11163

Table 2: Detection performance of machine learning models on lab dataset.

Feature	Accuracy	Precision	Recall	F1-score	AUC
GI [24]	0.9634	0.9802	0.9091	0.9431	0.9499
OP [23]	0.9660	0.9666	0.9321	0.9486	0.9577
Malconv [8]	0.9776	0.9957	0.9371	0.9653	0.9675
MBGINet-FCG	0.9732	0.9921	0.9312	0.9607	0.9636
MBGINet-CFG	**0.9803**	**0.9954**	**0.9453**	**0.9697**	**0.9716**

input to build a deep learning model for miner malware detection. These three baseline methods are representative research for miner malware and general malware detection in the static analysis.

Table 2 presents the performances of our proposed method and the baseline methods, where the miner malware detectors trained by the MBGINet method on two-level graph features are denoted as MBGINet-FCG and MBGINet-CFG, respectively. The bold values in Table 2 represent the best performance in this one column. It can be seen that MBGINet-CFG obtains the best results in every performance metric among all the detectors and achieves a significant advantage. For example, MBGINet-CFG achieves a result of 0.9803 in the accuracy metric, a 0.3 percentage point improvement over Malconv that is the best method among the baseline methods. Moreover, for a composite metric of F1-score, MBGINet-CFG achieves a 0.9 percentage point improvement over the Malconv method. On the other hand, MBGINet-FCG also achieves better results relative to the GI and OP methods. This illustrates the effectiveness of the MBGINet. Although the methods listed in Table 2 all obtained good detection results, the weakest GI method also obtained an accuracy of more than 0.96. However, such results do not directly reflect the test results of real-world data environments. To better approximate the performance of the miner detectors in the real world, we test the models' performances using a larger dataset to compare their generalization ability.

4.3. Performance Comparison on Simulated In-The-Wild Dataset. Once a miner detector is deployed in the wild, it often needs to face a large amount of data. This causes the range of feature variations of miner malware to increase significantly due to the artificial nature of malware. In this section, we choose Sim-Wild dataset as the performance test benchmark. Its data size is ten times larger than the test samples of lab dataset. By inspecting various miner detectors' performance on the Sim-Wild dataset and their changes relative to the lab dataset, we present the detection performance and generalization performance of different miner malware detectors. In the experiments, detectors trained in five cross-validation experiments were used as test subjects to examine corresponding methods performance on the simulated in-the-wild dataset. Moreover, the average of results in the five experiments was used to represent the performance of the tested method. Such a paradigm ensures machine learning methods are tested on the same models and datasets.

As in Table 2, the bold values in Table 3 represent the optimal performance in this column. In Table 3, MBGINet also achieves the leading position. The precision rate of MBGINet-CFG is higher than the second place of Malconv with 1.33 percentage points. Similar to Table 2, MBGINet-CFG still outperforms other methods and MBGINet-FCG is weaker than the Malconv model. Except for the Malconv model, the MBGINet is far superior to the other two specialized miner malware detection baselines. Opcode (OP) method [23] and grayscale image (GI) method [24] suffer a significant drop in all metrics. OP has the largest drop in recall rate, reaching 13.58 percentage points. This show that both machine learning methods do not learn universal features on the lab dataset, resulting in local optimization. MBGINet-CFG still outperforms the Malconv model in all metrics except the recall rate. MBGINet-CFG has a better precision rate but a weaker recall rate than Malconv, which indicates that MBGINet-CFG captures more essential features of miner malware and obtains a clearer picture of miner malware. The recall loss shows MBGINet-CFG does not cover nonmainstream miner malware behavior. Nevertheless, MBGINet-CFG outperforms Malconv in the other metrics. MBGINet-CFG's two comprehensive metrics are ahead of Malconv by 0.3 percentage points and 0.15 percentage points, respectively. This illustrates that MBGINet-CFG has comprehensive capabilities to detect miner malware.

The results in Tables 2 and 3 show that the MBGINet proposed in this paper has excellent performance on the task of miner malware detection. In the MBGINet, the behavior feature at the CFG level is significantly better than the behavior feature at the FCG level for the classification task. Both the GI features and opcode features achieve poor results for multiple reasons. The GI [24] may be because the browser bytecode file format is simpler than PE file format, so it is difficult for the CNN to model the characteristics of miner malware in PE format. The opcode's poor results may be because its sequential nature is inconsistent with the actual behavior sequence of miner malware, which contains a large number of jump actions.

In the experiments, Malconv obtained stronger performance than MBGINet-FCG. Such results are due to at least two reasons: on the one hand, the FCG features are still

Table 3: Detection performance of machine learning models on Sim-Wild dataset.

Feature	Accuracy	Precision	Recall	F1-score	AUC
GI [24]	0.9248	0.9561	0.8121	0.8783	0.8967
OP [23]	0.9146	0.9459	0.7963	0.8647	0.8863
Malconv [8]	0.9503	0.9720	**0.8766**	0.9218	0.9320
MBGINet-FCG	0.9411	0.9566	0.8533	0.9020	0.9176
MBGINet-CFG	**0.9556**	**0.9853**	0.8730	**0.9258**	**0.9335**

Table 4: Time consumption of methods for miner malware detection (in seconds).

Methods	Preprocess	Training stage	Inference stage
GI	0.008	$4.9810 * 18$	0.00150
OP	3.74	$626.08 * 16$	0.42089
Malconv	0	$205 * 7$	0.08858
MBGINet-FCG	3.63	$2.68 * 30$	0.00067
MBGINet-CFG	3.63	$7.8 * 10$	0.00326

too sparse with low feature dimensionality, resulting in the behavior pattern of miner malware not being well represented, and on the other hand, Malconv, as a variant of a traditional CNN model, has excellent learning performance and has been proven to have good modeling capabilities for static features of malware [13, 28]. For the MBGINet model, the model performance at the CFG level outperforms that at the FCG level, which is in line with our expectation because the underlying jumps at the basic block level are more diverse and have more call-and-call relationships. This allows the model's graph isomorphism learning capability to be better exploited.

However, although the method proposed in this paper achieves better results compared to the baseline methods, this paper also has the common drawbacks of static analysis-based malware detection methods, for example, packer obfuscation and other obfuscations against control flow analysis. These techniques can severely disrupt the behavior pattern under static analysis, causing MBGINet and other methods based on static features to fail. Fortunately, according to public reports [19], less than 30% of miner malware applied packer obfuscation, which shows that packer obfuscation has a limited impact on static analysis methods for miner malware detection.

4.4. Timely Consumption. We present the time consumption analysis of miner malware detection methods involved in this paper for feature preprocessing, model training, and sample inference. All experiments in this paper were performed on a computer with a NVIDIA 1080ti GPU and an Intel i7-7700 CPU. The experiments code was written in Python 3.7, and the Pytorch machine learning library with CUDA 10.1 is used as the deep learning framework.

As shown in Table 4, the "Preprocess" column indicates the computation time required for one sample. The "Training stage" column consists of epoch time and the number of epochs required. The time required for one sample inference is displayed in the "Inference stage" column.

From Table 4, it can be seen that our proposed MBGINet method has better time efficiency for both levels of behavior features. Although using the least time, GI feature's detection performance is poor. Malconv and biLSTM-OP methods need a mass of training time due to the huge number of parameters. In the MBGINet method, the computation speed of FCG features is better than that of CFG, because the structure of MBGINet-FCG applied is simpler. Putting the above analysis together, the MBGINet method still has advantages in time efficiency for training and testing, especially compared with the Malconv and OP, where the training speed is greatly reduced.

5. Conclusion

The behavior pattern of miner malware is distinctly different from other malware and benign software due to the apparent CPU computing and auxiliary malicious behavior characteristics. In this paper, a miner malware detection method MBGINet is designed based on behavior pattern recognition. MBGINet is designed with aggregator layers and fully connected layers to model the connectivity characteristics between nodes in function call graph or control flow graph. For FCG and CFG features, MBGINet has different network structures to better model node convergence structures. Experimental verification shows the MBGINet can accurately describe the behavior pattern of miner malware compared to the opcode feature [23]. Although MBGINet uses similar static behavioral features as opcode, it achieves a 3.08% accuracy improvement over the opcode method. Moreover, experiment results show that the MBGINet method achieves 4.1% accuracy gains relative to the grayscale image feature [12], which is used to detect browser cryptohijack byte files [24]. Besides, MBGINet-CFG is also better than the popular malware detection method of Malconv [8] in the comprehensive metrics. These experimental results show that the MBGINet is an effective behavioral pattern analysis method for miner malware detection in static analysis. Experiments under simulated in-the-wild scenarios also demonstrate the performance stability of MBGINet, indicating that MBGINet is more practical than other baseline methods.

Data Availability

The datasets used in this paper are publicly available and fully desensitized, without any privacy or security risks. The public datasets are available at https://datacon.qianxin.com/opendata/maliciouscode. Note: Following the policy of data provider, these datasets are currently open for research needs/teaching needs. Therefore, it is necessary to submit a request file with information about the research organization to the owner of the datasets copyright. For more information, please refer to the abovementioned webpage.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

First of all, the authors would like to thank QI-ANXIN Technology Group Inc. for organizing the Big Data Security Analysis Contest 2020 (DataCon2020) and sharing the cryptocurrency mining malware dataset. This work was supported by the National Natural Science Foundation of China (Grant nos. 61972297, 62172308, and 62172144) and National Key Research and Development Program of China (Grant no. 2018YFB0805005).

References

[1] Randi Eitzman BWJK Kimberly Goody, "How the rise of cryptocurrencies is shaping the cyber crime landscape: the growth of miners," 2021, https://www.fireeye.com/blog/threat-research/2018/07/cryptocurrencies-cyber-crime-growth-of-miners.html.

[2] McAfee, "McAfee labs threats report," 2021, https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-apr-2021.pdf.

[3] E. Tekiner, A. Acar, A. S. Uluagac, E. Kirda, and A. A. Selcuk, "Sok: cryptojacking malware," 2021, https://arxiv.org/abs/2103.03851.

[4] "360TS. Cryptominer, winstarnssmminer, has made a fortune by brutally hijacking computers," 2021, https://blog.360totalsecurity.com/en/cryptominer-winstarnssmminer-made-fortune-brutally-hijacking-computer/.

[5] R. A. Bridges, S. Oesch, M. E. Verma et al., "Beyond the hype: a real-world evaluation of the impact and cost of machine learning–based malware detection," 2020, https://arxiv.org/abs/2012.09214.

[6] Q. Dong, C. Jia, F. Duan, and D. Wang, "RLS-PSM: a robust and accurate password strength meter based on reuse, Leet and separation," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4988–5002, 2021.

[7] W. Melicher, B. Ur, S. M. Segreti et al., "Fast, lean, and accurate: modeling password guessability using neural networks," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pp. 175–191, Austin, TX, USA, August 2016.

[8] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," 2017, https://arxiv.org/abs/1710.09435.

[9] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," 2018, https://arxiv.org/abs/1804.04637.

[10] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining-KDD'04*, Seattle, WA, USA, August 2004.

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

[12] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security-VizSec'11*, pp. 1–7, Pittsburgh, PA, USA, July 2011.

[13] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," in *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW)*, pp. 21–27, IEEE, San Francisco, CA, USA, May 2019.

[14] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, and M. McLean, "Classifying sequences of extreme length with constant memory applied to malware detection," 2020, https://arxiv.org/abs/2012.09390.

[15] R. Vyas, X. Luo, N. McFarland, and C. Justice, "Investigation of malicious portable executable file detection on the network using supervised learning techniques." in *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 941–946, IEEE, Lisbon, Portugal, May 2017.

[16] S. Jeon and J. Moon, "Malware-detection method with a convolutional recurrent neural network using opcode sequences," *Information Sciences*, vol. 535, pp. 1–15, 2020.

[17] J. Yan, G. Yan, and D. Jin, "Classifying malware represented as control flow graphs using deep graph convolutional neural network," in *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019*, pp. 52–63, Portland, OR, USA, June 2019.

[18] A. A. E. Elhadi, M. A. Maarof, B. I. A. Barry, and H. Hamza, "Enhancing the detection of metamorphic malware using call graphs," *Computers & Security*, vol. 46, pp. 62–78, 2014.

[19] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem," in *Proceedings of the Internet Measurement Conference*, pp. 73–86, Amsterdam, Netherlands, October 2019.

[20] A. Zimba, Z. Wang, M. Mulenga, and N. H. Odongo, "Crypto mining attacks in information systems: an emerging threat to cyber security," *Journal of Computer Information Systems*, vol. 60, no. 4, pp. 297–308, 2020.

[21] A. Pastor, A. Mozo, S. Vakaruk et al., "Detection of encrypted cryptomining malware connections with machine and deep learning," *IEEE Access*, vol. 8, pp. 158036–158055, 2020.

[22] F. Gomes and M. Correia, "Cryptojacking detection with cpu usage metrics," in *Proceedings of the 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, pp. 1–10, IEEE, Cambridge, MA, USA, November 2020.

[23] A. Yazdinejad, H. HaddadPajouh, A. Dehghantanha, R. M. Parizi, G. Srivastava, and M.-Y. Chen, "Cryptocurrency malware hunting: a deep recurrent neural network approach," *Applied Soft Computing*, vol. 96, Article ID 106.630, 2020.

[24] F. Naseem, A. Aris, L. Babun, E. Tekiner, and A. S. Uluagac, "MINOS: a lightweight real-time cryptojacking detection system," in *Proceedings of the 2021 Network and Distributed System Security Symposium*, pp. 21–25, San Diego, CA, USA, November 2021.

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2018, https://arxiv.org/abs/1810.00826.

[26] S. Yue, "Imbalanced malware images classification: a CNN based approach," 2017, https://arxiv.org/abs/1708.08042.

[27] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460-461, pp. 83–102, 2018.

[28] S. Bose, T. Barao, and X. Liu, "Explaining ai for malware detection: analysis of mechanisms of malconv," in *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, Glasgow, UK, July 2020.