IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 6, 2020, accepted February 29, 2020, date of publication March 4, 2020, date of current version March 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2978335

Exploring Function Call Graph Vectorization and File Statistical Features in Malicious PE File Classification

YIPIN ZHANG[1], XIAOLIN CHANG[1], (Member, IEEE), YUZHOU LIN[1], JELENA MIŠIĆ[2], (Fellow, IEEE), AND VOJISLAV B. MIŠIĆ[2], (Senior Member, IEEE)

[1]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China
[2]Computer Science Department, Ryerson University, Toronto, ON M5B 2K3, Canada

Corresponding author: Xiaolin Chang (xlchang@bjtu.edu.cn)

The work of Yipin Zhang, Xiaolin Chang, and Yuzhou Lin was supported in part by the National Natural Science Foundation of China under Grant U1836105, and in part by the National Key Laboratory of Science and Technology on Information System Security. The work of Jelena Mišić and Vojislav B. Mišić was supported in part by the National Science and Engineering Research Council of Canada (NSERC) through Discovery Grants.

ABSTRACT Over the last few years, the malware propagation on PC platforms, especially on Windows OS has been even severe. For the purpose of resisting a large scale of malware variants, machine learning (ML) classifiers for malicious Portable Executable (PE) files have been proposed to achieve automated classification. Recently, function call graph (FCG) vectorization (FCGV) representation was explored as the input feature to achieve higher ML classification accuracy, but FCGV representation loses some critical features of PE files due to the hash technique. This paper aims to further improve the classification accuracy of FCGV-based ML model by applying both graph and non-graph features. We propose an FCGV-SF based Random Forest classification model, which applies both FCGV features (graph features) and statistical features (SF, non-graph features) extracted from disassembled PE files. Six types of effective non-graph features are chosen for our integrated vector, namely, metadata, symbol, operation code, register, section and data definition. We evaluate our model on a dataset provided by Microsoft hosted at Kaggle, and the experimental results indicate that the classification accuracy increases from 0.9851 to 0.9957 compared with the existing model based on FCGV only.

INDEX TERMS Function call graph, machine learning, malware classification, Portable Executable, statistical features.

I. INTRODUCTION

Over the last few years, the malware propagation on PC platforms, especially on Windows OS, has been even severe, causing various threats to system security and data privacy. AV-TEST statistical report [1] indicated that there were over 30 million malicious Portable Executable (PE) files registered merely in the first half of 2019. This huge intrusion of malicious PE files results from the malware modification and obfuscation performed by attackers, so that similar malware samples will be distinct from the others [2] and then evade detection.

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

There are usually two separate steps to perform on each malicious PE file, namely malware detection and classification. Firstly, if an executable program contains any malicious content, it needs detecting through malware analysis techniques. After malware detection, a classification mechanism categorizes the executable program labeled as malware into the most similar family for further analysis. In this paper, we focus on the techniques of malware classification. To extract features used for malware classification, static and dynamic analysis techniques have been explored to perform the analysis of malicious PE files. Static analysis examines the codes of malware samples without executing them. In static analysis, content-based features such as instruction opcodes [3], API sequences [4], [5] and function call graphs

(FCGs) [6], [7] are typically extracted from disassembled malicious PE files as the original features for analysis. Static analysis can easily capture syntax and semantic information for in-depth analysis, but it is susceptible to code obfuscation techniques, e.g., compression and polymorphic/metamorphic transformation [8]. Dynamic analysis usually executes malware samples in a virtual environment which is monitored by debugger [12] for observing their behavioral information such as network activities [9], system calls [10], file operations and registry modification records [11]. Code obfuscation technologies exert less effect on dynamic analysis, but malware execution consumes much more time and many more resources than static analysis. Both static and dynamic analysis techniques have their unique strength and weakness.

A large number of PE malware variants poses great challenges to human experts in manually analyzing all of these malware. This situation exposes an imperative need for developing effectively and efficiently automated malware classification techniques. Using machine learning (ML) in the malware classification can make a significant contribution to resist the malware epidemic. ML classifiers used for malicious PE file classification typically employ a single numerical feature vector representation of each file as input and mark one or more class labels for each file during training. By performing static and/or dynamic analysis on each PE sample, two types of features can be extracted from malware binaries, namely, non-graph features and graph features. Recently function call graph (FCG) vectorization (FCGV), which is a kind of graph features, was explored to achieve higher ML classification accuracy [13] but FCGV representation loses some critical features of PE files due to the hash technique. Meanwhile, non-graph features have been applied for malware classification [14]. Each type of features represents its unique perspective of malicious PE files, having its own merits and limitations. Hence, it is a necessity to creating an integrated feature vector which contains more comprehensive information of PE binaries. However, there is no work on the integration of FCGV features and non-graph features for designing ML malware classifiers.

In this paper, we propose an FCGV-SF based Random Forest classification model (denoted as FCGV-SF model in the following) which applies both FCGV features (graph features) and statistical features (SF, non-graph features) extracted from disassembled PE files. Statistical features reflect the high-level statistical characteristics in PE binaries, which is more concise and representative. Six types of effective statistical features [14] are chosen to build our integrated vector, namely metadata, symbol, operation code, register, section and data definition. To the best of our knowledge, we are the first to apply both FCGV features and non-graph features for malware classification. Compared with prior malware classification work based on FCGV only or non-graph statistical features only, our proposed model preserves more vital information in disassembled PE files. We use the data provided on Kaggle [2] for Microsoft Malware Challenge to evaluate our model, and the classification accuracy increases from 0.9851 to 0.9957 by comparison with the existing model based on FCGV only.

The rest of this paper is divided into four sections as follows. Section II discusses related work. Section III presents the details of our FCGV-SF model using the new integrated vector. Section IV presents experimental results and Section V concludes this study and describes future work.

II. RELATED WORK

The past years witnessed various ML-based approaches, most of which depended on the features extracted from malware binaries by using static and/or dynamic analysis. These features can be organized as two groups, one is graph feature and the other is non-graph feature. We discuss the existing classification approaches from the aspects of graph and non-graph features in the following.

A. GRAPH FEATURE BASED APPROACHES

There are usually three main types of graph information from malware samples: FCGs, system-call dependency graphs and control flow graphs. An FCG is a directed graph representation constructed from codes where the vertices specify functions and the edges correspond to the caller-callee relations between functions (vertices) [20]. A system call dependency graph is a directed graph that is usually determined by dynamic taint analysis. In a system call dependency graph, a vertex corresponds to a system call and an edge represents a data dependency between system calls. In [21], Allen defined a control flow graph as a ''directed graph where basic code blocks are represented by vertices and control flow paths are represented by edges''. A basic control block was described as ''a linear program instructions sequence which has one entry point (the first instruction executed) and one exit point (the last instruction executed)''. Graph-based features have been increasingly used in many researches to cluster and classify malware. Such features have the most significant advantages of preserving interactive information between different parts of the malicious codes.

This section only discusses features extracted by static analysis, which are called as FCGs. FCGs are usually built from disassembled binaries constructed by static analysis. Various researches have extracted FCG features for malware classification and clustering. After creating FCGs, we need a measure to evaluate the similarity between two FCGs, such as approximate graph edit distance (GED). In [22] and [23], Simulated Annealing Algorithm [24] was employed to approximate GED. On the other hand, Hu *et al.* [25] used Hungarian Algorithm to approximate GED. Hassen and Chan [13] developed a function clustering-based FCGV representation using hash technique to approximate GED, which achieved remarkable performance as well as improved classification accuracy.

Note that GED is not the only way to measure graph similarity. For example, as another measure of similarity, the normalized common edge number between two graphs was used in [26]. Dullien and Rolles [27] computed graph similarity through fixed points and propagations.

They defined a fixed point between pair of graphs. A fixed point represents two nodes (one each from two different graphs), which can be easily determined to represent the same item in both graphs. Their method started from an initial fixed point. By considering the adjacent nodes, it propagated to more fixed points. In addition, Kong and Yan [28] extracted new features from FCGs and used these features to approximate the similarity between two graphs.

All these aforementioned works only applied FCG features. FCGs have an advantage on preserving more complete structural information in binaries, however, loss of some structural information is inevitable during the extraction. This representation fails to extract comprehensive features of code, which leads to suboptimal classification accuracy. It highlights the necessity to integrate non-graph features as well as FCGs. Hence, our model explores both FCG and non-graph features for classification.

B. NON-GRAPH FEATURE BASED APPROACHES

This section presents research work based on common non-graph features which can be extracted from one malicious PE file often with less preprocessing than graph features.

Ahmadi *et al.* [14] provided a constructive set of non-graph features. They extracted several customized statistical features which reflected the essence of maliciousness in disassembled PE files. For example, the symbols like "−, +, ∗,], [, ?, @" are typical of code that has been designed to evade detection. Jung *et al.* [15] explored the application of function lengths in bytes, which were from disassembled malware samples and represented as a histogram with the predefined number of bins. Then they used these frequency values of function lengths as features for the malware classification. In [16] and [17], another set of features based on printable strings were extracted using static analysis from malware binaries. Besides, the authors in [17] and [18] used program import table to create a binary feature vector for malware classification. The program import table in a PE file can provides information for the program so that it can import the function in external libraries while executing.

Instruction n-gram constructed from instruction sequences are extracted from a disassembled malicious PE file [18], [19]. In this case, instruction mnemonics or opcodes, excluding the operands, can represent the instructions. Hu *et al.* [19] emphasized that the high dimensionality of instruction n-gram was one of the challenges when using them. Even for 2-gram features, the number of 2-grams can possibly reach tens of thousands. To address this problem, a hashing trick was used to reduce dimensionality. They employed a uniformly distributed hash function on the large feature space to hash this high-dimension space into a smaller dimension. In [17], binary file byte sequence n-gram was used as features for malware classification. Unlike n-gram sequences based on instruction mnemonic or opcodes, these features do not require disassembled files. These byte sequences are extracted from all sections of the original PE binaries, which differ from the opcode sequences that only focus on the code segments.

All these works only applied non-graph features. Although it is mostly easy for us to extract non-graph features, it still turns out to be hard to remain comprehensive information from binaries. As a result, we combine non-graph features with FCG features to achieve more accurate classification results.

III. FCGV-SF BASED RANDOM FOREST CLASSIFICATION MODEL

This section first presents the overview of our FCGV-SF model, then details the procedure of FCG vectorization and statistical feature extraction respectively.

A. FCGV-SF OVERVIEW

Selecting features which represent malware samples is one of the most challenging issues for the classification task. This paper combines various features (namely, FCGV representation and non-graph statistical features) to generate a novel integrated feature vector in order to attain better classification accuracy.

We choose the features extracted from FCGs as the first part because they preserve more complete structural information of codes, compared with n-gram features. They include the information of functions in a malicious PE file, and more importantly contain the interactive information between functions. However, most of the existing researches, which employed FCG-based features for malware classification, relied on computationally intensive techniques to estimate graph similarity. As a result, these work exposed shortcomings of large performance overhead and weak scalability. Hence, we use FCGV technique [13] to reach more accurate results with less time cost. The details are discussed in Subsection III.B.

Note that non-graph features, compared with graph features, are easily extracted and they are also helpful in improving classification accuracy. Meanwhile, FCGV representation can be easily combined with other non-graph features. In the midst of various non-graph features, we investigate a set of statistical features [14], namely intuitively statistical characteristics of PE files, such as file size and the number of lines in the file. The main attraction of statistical features is that they represent global characteristics of PE files which is more concise and representative as well. To avoid unnecessary performance overheads, we devise a simple, yet efficient feature extraction module without using more complex features based on n-grams or sequences. We eventually choose six types of statistical features including metadata, symbol, operation code, register, section, data definition, as discussed in Subsection III.C.

Fig.1 presents a high-level view of our classification model. As it shows, the first step of our model is the extraction of FCG representations from disassembled malicious PE binaries. Once an FCG is extracted, we use function clustering techniques for vectorizing this FCG to an FCGV vector in a low-dimensional feature space, which is defined

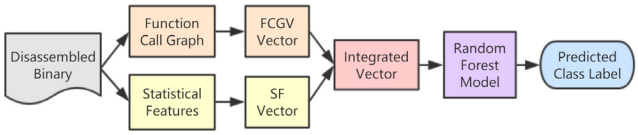


FIGURE 1. The overview of FCGV-SF model.

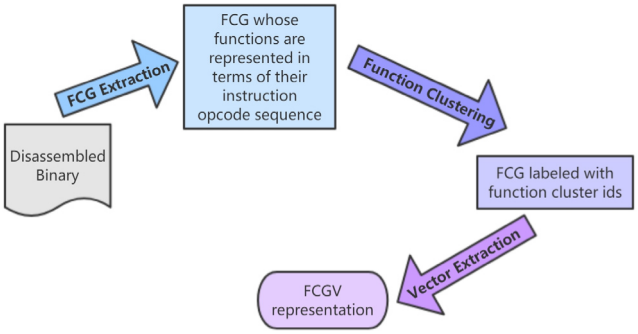


FIGURE 2. The procedure of FCG vectorization.

as $\mathbf{v}_{FCGV}$. Meanwhile, we collect six types of predefined statistical features to build a non-graph SF vector $\mathbf{v}_{SF}$. Then we concatenate $\mathbf{v}_{FCGV}$ and $\mathbf{v}_{SF}$ to create an integrated feature vector $\mathbf{v}$. At last, malware classification is performed on $\mathbf{v}$through the ML model based on Random Forest Algorithm, which predicts the malware family that the sample belongs to. We discuss Random Forest classifier in Subsection III.D.

B. FCG VECTORIZATION

The procedure of converting an FCG to FCGV representation is presented in Fig.2. Three modules are involved in FCG vectorization, including FCG Extraction, Function Clustering and Vector Extraction. Our model starts by first performing the extraction of FCGs from disassembled binaries. Vertices in FCGs are functions represented in terms of their instruction opcode sequences. Then the hash technique for clustering functions reconstructs an FCG labeled with function cluster-ids. The final step is to perform vector extraction on this FCG into FCGV representation. Details of these three modules are discussed in the following.

1) FCG EXTRACTION

FCG Extraction is the first module of FCGV technique which extracts FCG representations from disassembled binaries of malicious PE files. In order to obtain the functions and their caller-callee relations, we use the regular expression to match instruction opcodes in binaries. During the process of FCG Extraction, vertices which represent external functions are labeled with their function names. Note that the names of local functions named by malware writers generally cannot reflect what the function implements with their instruction sequences. Therefore, we decide to label later the vertices which represent the local functions. In this FCG, vertices corresponding to local functions also contain a set of instruction opcode sequences extracted from these functions. Besides, the caller-callee relations between functions are defined as directed and unweighted edges. When FCG Extraction is finished, this labeled graph is passed to the next module which is responsible for clustering and relabeling local functions.

2) FUNCTION CLUSTERING

The second module of vectorization is Function Clustering, which needs a proper measure to identify similar functions. GED is one of effective ways to estimate similarity, however, with $O(n^2)$ time complexity in the number of instructions. To address this problem, Jaccard Index can be employed to approximate GED but it still costs much time. Fortunately, locality-sensitive hashing (LSH) is an algorithmic technique that preserves relative distances between items while hashing similar input items into low-dimension versions. Hence, it can effectively reduce the dimensionality of high-dimension data. A set of LSH functions called Minhash [30] can be used to efficiently approximate Jaccard Index, which further simplifies the measurement of similarity between functions. The procedure of Minhash is as follows. First we employ a random permutation on the set. Then the index value of the first element is computed. Guided by the rationale of Minhash, this value is the hash of the set. Under the strict mathematic verification, the probability that two sets have the same Minhash value is equivalent to the Jaccard similarity between the two sets. Therefore, for the local function, we perform Minhash signature and secondary hashing on its n-gram instruction opcode set to compute a positive cluster-id. For the external function, we directly hash the function name to a negative cluster-id. Then we get an FCG whose functions are labeled with cluster-ids.

3) VECTOR EXTRACTION

In the third module, we extract FCGV representation from the FCG labeled with function cluster-ids. Vertex weight and edge weight constitute this vector representation. In this labeled FCG, the number of functions (vertices) from each cluster is represented by the vertex weight. As for the edge weight, it corresponds to the number of times that a caller-callee relation is found from a function (vertex) in one cluster to a function (vertex) of another cluster or a function (vertex) within the same cluster. We can create an FCGV representation of FCG by concatenating the weights of vertices and edges.

Due to function clustering, two graphs that are slightly different may have the exactly same FCGV representation. In the area of malware classification, there are various obfuscation techniques such as changing function calling patterns. This FCGV representation has its own merit which is more resilient to resist the negative effects of these techniques. The reason is that it might not express small changes in the graph structure on condition that there is no variation observed in the edge and vertex frequencies. Therefore, FCGV representations make the malware classification less susceptible to malicious changes such as function calls reordering. However, the original FCG inevitably loses a few details of the graph structure after being processed

TABLE 1. Six types of statistical features.

Name	Description	Selected Reason
Metadata (MD2)	The size of the file and the number of lines in the file	Most intuitive features
Symbol (SYM)	The frequencies of the following set of symbols, - , +, *,], [, ?, @	Frequent use of these characters in code is a typical design of evading detection, for instance by having recourse to indirect calls, or dynamic library loading
Operation Code (OPC)	The frequencies of 93 operation codes that are frequently used	Operation codes represent machine code by mnemonic to symbolize assembly instruction
Register (REG)	The frequency of the use of registers	Most of the x86 processor registers are used for particular tasks, but programmers rename registers in some cases to make the analysis harder
Section (SEC)	The proportion of predefined sections like .text, .data, .rdata, .edata, .idata,.bss, .rsrc, .tls, and .reloc	Using evasion techniques like packing, a programmer can modify and reorder the default sections even create the new sections
Data Definition (DP)	The proportion of db, dw, and dd instructions	Because of packing, some malware samples just include some of these instructions instead of any API call

by the hash technique. Hence, we combine this FCGV representation with statistical features to make up the loss.

C. STATISTICAL FEATURES EXTRACTION

FCGV representations only use code segments in binaries. Furthermore, they are simplified through the Minhash technique, which means they do not preserve all the critical information from disassembled files. As a result, it leads to the loss of expressiveness and suboptimal classification accuracy. In this subsection, we consider combining statistical features with FCGV representations.

Then we face the challenge of how to choose proper statistical features for the integrated vector. On the one hand, we need to ensure that FCGV vector and SF vector are of equal importance, which means we should make the dimension of $\mathbf{v}_{FCGV}$ and $\mathbf{v}_{SF}$ basically identical. On the other hand, we choose the effective and representative statistical features in terms of the experiments in [14].

We notice that the dimension of API (one set of statistical features) is too high and MISC (another set of statistical features) includes too many items that only have auxiliary effects. Therefore, we abandon these two sets. In Table 1, we provide details on what features we have chosen and why they have been selected. Extracting these six types of features from disassembled files, we combine them with the FCGV vector presented in Subsection III.B to create a new feature vector as the input of our ML model. Not only does this vector contain structural information in the code, but also it has intuitively statistical characteristics of binaries. The more effective features it can preserve, the higher accuracy it may lead to.

D. RANDOM FOREST CLASSIFIER

Random Forest [29] is a ML algorithm using ensemble learning whose base classifiers are decision trees. Ensemble learning techniques generally combine predictions from multiple classifiers to improve prediction accuracy. Random Forest Algorithm starts from building T decision trees. We randomly select training samples for each tree from the original training dataset D to get a new training set D_t which has the same size with the original D. This random selection is based on some certain distribution. Then we train an individual decision tree on D_t without pruning. For the split of any given tree node, we only consider F features which are randomly selected rather than all available features. Finally, the majority vote policy is adopted among all the trees to predict the class label of the sample.

Using Random Forest Algorithm for malware classification brings us various advantages [31] as follows.

- Random Forest tends to have a better prediction performance on a large-scale set of features because it allows us to construct individual trees whose decorrelation improves the classification accuracy.
- Random Forest has a fast training speed, since for each tree node, it considers a randomly selected subset of features which is much smaller than our entire feature space. Hence, it results in a training time reduction.
- Random Forest can prevent overfitting to a certain extent, since the training samples for each tree and the features for the split of each tree node are randomly selected. Besides, because of the Law of Large Numbers, Random Forest does not overfit as more trees are added [29].
- Random Forest reaches better or equal prediction accuracy compared with other techniques, such as decision tree, logistic regression, and backpropagation artificial neural networks [32].

In our FCGV-SF model, the integrated vector $\mathbf{v}$ consists of two parts, $\mathbf{v}_{FCGV}$ and $\mathbf{v}_{SF}$, whose dimensions depend on optimal parameters chosen in FCG vectorization. The integrated vector $\mathbf{v}$ is given as input to Random Forest classifier and the output is a predicted label that represents which family the sample belongs to.

IV. EXPERIMENTAL EVALUATION

This section evaluates the capability of our FCGV-SF model. We first present the dataset and then determine the optimal values of parameters, namely the n-gram length, the number of clusters and the number of decision trees. Eventually we show the improvement of our FCGV-SF model compared with the previous FCGV-only model.

A. DATASET

Microsoft Malware Classification Challenge (BIG 2015) dataset [2] hosted at Kaggle is used for evaluating our

TABLE 2. Microsoft malware dataset class distribution.

Malware Family Name	Number of Samples
Ramnit	1513
Lollipop	2470
Kelihos ver3	2936
Vundo	446
Simda	34
Tracur	294
Kelihos_ver1	387
Obfuscator.ACY	1168
Gatak	1012

proposed classification model. The reasons that we choose this dataset are presented as follows.

- Suitable. This paper focuses on malware classification. To the best of our knowledge, other malware datasets only label benign and malicious samples, like Ember (2018) [33], which cannot be used to classify malware variants into their actual families due to the lack of specific class labels. Fortunately, this Kaggle dataset labels nine specific families of malicious PE files, which means this dataset exactly meets our requirement.
- Comparable. The FCGV-only model proposed in [13] used this Kaggle dataset for evaluation. To demonstrate the improvement of our FCGV-SF model, we choose the same dataset to intuitively compare the classification accuracy.
- Convenient. This Kaggle dataset includes disassembled files constructed by performing static analysis on original malicious programs. These disassembled files can be directly used for feature extraction. Besides, each sample in this dataset is marked with their class label, which can verify the classification results of our FCGV-SF model.

There are 10,867 labeled malware samples in this original dataset. We use 10,260 of the samples that are successfully parsed by the disassembled file parser proposed in [13]. Table 2 shows the class distribution of these samples. Each malware sample involves two files, one including the disassembled code and the other one consisting of the hex code. We fail to extract statistical features from the hex code in Kaggle dataset because of the code format. Thus the disassembled code is used for experiments in this section.

B. PARAMETER SELECTION

In our FCG vectorization, there are two parameters to be experimentally configured, namely, the instruction opcode n-gram length and the function (vertex) cluster number. When local functions are represented as sets of n-gram sequences, the length of n-gram determines the value of n in n-gram, which denotes the number of instruction opcodes. The number of function clusters determines how many buckets that functions are hashed into.

Fig.3 shows that the impact of both the n-gram length and the function cluster number on the classification accuracy. We observe that the accuracy of using 1-gram is better than that of both 3-gram and 5-gram whatever the number of clusters is. LSH functions used for GED approximation can

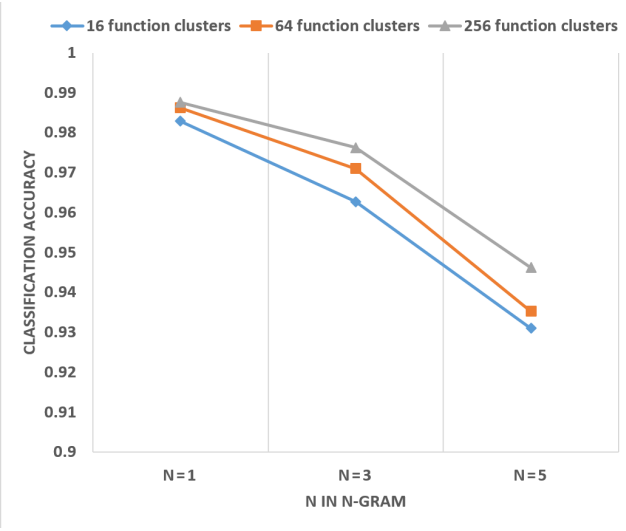


FIGURE 3. Effect of n-gram length and function cluster number on classification accuracy.

demonstrate this result. We employ the method proposed in [13], which uses a set of LSH functions called Minhash to compute Jaccard Index for the approximation of GED. When calculating GED, the operations of insertion and deletion are performed on 1-gram. Therefore, when we approximate GED with Minhash, the classification accuracy will be worse under the condition of the longer n-gram. From Fig.3, we observe that:

- When it comes to the function cluster number, the increase of function clusters results in the improving classification accuracy. It can be explained by the fact that if we cluster functions into a smaller number of groups, the possibility of hashing dissimilar functions into the same cluster becomes higher. Obviously, this leads to less accurate classification results in turn.
- However, the increase of function clusters does not have significant influence on the accuracy improvement when using 1-gram. Note that a large number of function clusters will result in a high-dimension vector space and unnecessary performance overhead. Therefore, our optimal model is based on 1-gram opcode sets and 16 function clusters.

Besides, in our Random Forest classifier, we need to determine the optimal parameter namely the optimal number of decision trees. As the number of trees increases from 10 to 100, we evaluate the classification accuracy and weighted F1 score with 5-fold cross validation on the FCGV-only model proposed in [13] as well as our FCGV-SF model. Weighted F1 score is the harmonic mean of precision and recall weighted by the number of samples for each family. From Table 3, we find that the increase of trees does not improve the classification accuracy and weighted F1 score obviously when the number of trees reached 80. Note that with the increasing number of trees, classification time increases. To balance the classification time and the accuracy, we choose 80 as the optimal number of decision trees.

TABLE 3. Effect of decision tree number on classification accuracy and weighted F1 score.

	FCGV-only		FCGV-SF	
Number of Decision Trees	Accuracy	Weighted F1 Score	Accuracy	Weighted F1 Score
10	0.9791	0.9788	0.9941	0.9941
20	0.9835	0.9831	0.9953	0.9953
30	0.9836	0.9832	0.9955	0.9955
40	0.9841	0.9837	0.9956	0.9956
50	0.9845	0.9841	0.9957	0.9957
60	0.9847	0.9844	0.9957	0.9957
70	0.9846	0.9842	0.9957	0.9957
80	0.9851	0.9848	0.9957	0.9957
90	0.9851	0.9848	0.9958	0.9958
100	0.9853	0.9849	0.9959	0.9959

C. CLASSIFICATION ACCURACY COMPARISON

We evaluate our model by making a comparison with the classification model only based on FCGV proposed in [13]. As discussed above, they converted FCGs to FCGV representations using function clustering. This method has the main attraction of linear time cost and better performance. Besides, the extracted graph feature vector can be integrated with other non-graph features easily.

Our implementation of their model contains three modules, Function Clustering, Call Graph Vector Extraction, and Base Classifier. Functions (vertices) are firstly represented with 1-gram opcode sequences and then hashed into 16 clusters. The confusion matrix of classification on the entire dataset with 5-fold cross validation is shown in Fig.4, which achieves an overall accuracy of 0.9851. The decimals on the diagonal of the confusion matrix reflects the recall of each malware family. However, it is obvious to see that only a family named Simda does not reach the enough perfect recall. The reason is that there are only 34 Simda instances in the entire dataset for training. But from another point of view, the less perfect recall exposes that using FCGs alone will lose some representative features in malicious PE binaries.

Then we evaluate our FCGV-SF model on the same 10260 samples by predicting the labels on them. We convert each FCG to a 240 dimensional feature vector $\mathbf{v}_{\mathrm{FCGV}}$ by hashing functions into 16 clusters. In addition, we extract six types of statistical features to create a 170 dimensional feature vector $\mathbf{v}_{\mathrm{SF}}$. Through concatenating $\mathbf{v}_{\mathrm{FCGV}}$ and $\mathbf{v}_{\mathrm{SF}}$, we obtain an integrated vector $\mathbf{v}$ as the input of Random Forest classifier. The dimensions of $\mathbf{v}_{\mathrm{FCGV}}$ and $\mathbf{v}_{\mathrm{SF}}$ are basically identical, so these two kinds of features are of equal importance when feature subsets are randomly selected in Random Forest. The classification results of our model on the entire dataset with 5-fold cross validation is presented in Fig.5. Compared with the previous FCGV-only model, the overall accuracy increases from 0.9851 to 0.9957. In particular, the recall for the malware family called Simda improves from 0.41 to 0.97. In addition, the recall of other malware families also has a little improvement. These results indicate that statistical features can better describe the characteristics of malware

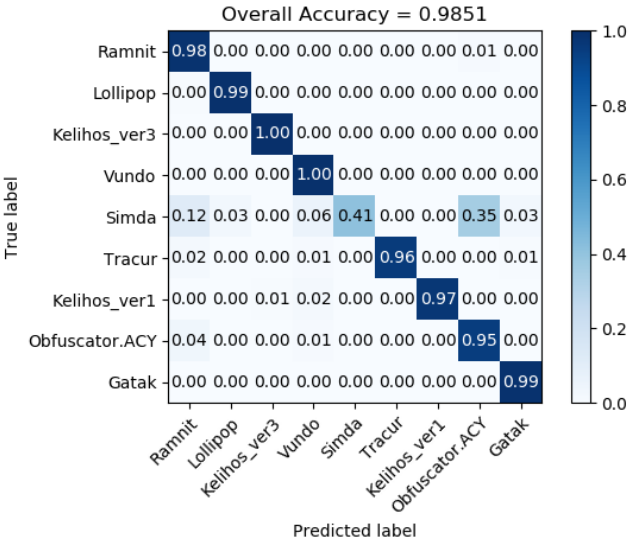


FIGURE 4. Confusion matrix of classification using FCG-only model.

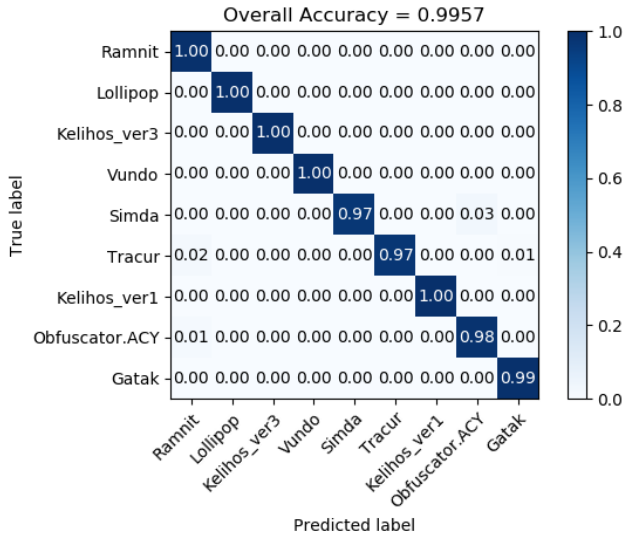


FIGURE 5. Confusion matrix of classification using FCGV-SF model.

TABLE 4. Evaluation on FCGV-only and our FCGV-SF model.

Model	Average Precision	Average Recall	Weighted F1 Score
FCGV-only	0.9852	0.9851	0.9847
FCGV-SF	0.9957	0.9957	0.9957

samples from Simda and other families, which highlights the necessity of integrating FCGV and statistical features.

Besides, we evaluate the FCGV-only model proposed in [13] and our FCGV-SF model in terms of average precision, average recall and weighted F1 score, which are weighted by the number of samples from each family. From Table IV, we observe that each of these three metrics has an improvement so as to prove the effectiveness of our model.

V. CONCLUSION AND FUTURE WORK

This paper explores the improvement in the classification accuracy of FCGV-only model. We propose an FCGV-SF based Random Forest classification model. The input of this

model is a novel feature vector which concatenates FCGV representation with non-graph statistical features. We select the optimal parameters (namely the n-gram length, the number of clusters and the number of decision trees), which can achieve a balance between time cost and accuracy for vectorizing an FCG to FCGV representation. Experiments are carried out for verifying the effectiveness of our model by comparing with FCGV-only classification results. Our proposed model is able to capture more information of malicious PE files by integrating more representative features together and then leads to a higher classification accuracy.

Note that the proposed model is evaluated by a malicious PE file dataset constructed from static analysis. It can only extract features in malicious codes, but a number of functions in binaries are only used for obfuscation instead of running in the execution. Hence, features are less comprehensive without analysis based on malware behaviors. Besides, this paper only focuses on malware classification. However, malware detection is also a challenging and important task. Therefore, we next plan to evaluate our model on a dataset which is not only constructed by static analysis as well as dynamic analysis but also contains benign and malicious PE files both.

REFERENCES

[1] AV-TEST. (Jul. 2019). *Malware Statistics & Trends Report*. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[2] Kaggle. (Apr. 2015). *Microsoft Malware Classification Challenge (BIG)*. [Online]. Available: https://www.kaggle.com/c/malware-classification

[3] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3838–3845.

[4] A. Pektaş and T. Acarman, "Classification of malware families based on runtime behaviors," *J. Inf. Secur. Appl.*, vol. 37, pp. 91–100, Dec. 2017.

[5] A. Pektaş and T. Acarman, "Malware classification based on API calls and behaviour analysis," *IET Inf. Secur.*, vol. 12, no. 2, pp. 107–117, Mar. 2018.

[6] H. Jiang, T. Turki, and J. T. L. Wang, "DLGraph: Malware detection using deep learning and graph embedding," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2018, pp. 1029–1033.

[7] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "A novel graph-based approach for IoT botnet detection," *Int. J. Inf. Secur.*, to be published.

[8] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 421–430.

[9] J. Jiang, Q. Yin, Z. Shi, and M. Li, "Comprehensive behavior profiling model for malware classification," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 129–135.

[10] T. Lee, B. Choi, Y. Shin, and J. Kwak, "Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient," *J. Supercomput.*, vol. 74, no. 8, pp. 3489–3503, Dec. 2015.

[11] G. Cabau, M. Buhu, and C. P. Oprisa, "Malware classification based on dynamic behavior," in *Proc. 18th Int. Symp. Symbolic Numeric Algorithms Sci. Comput. (SYNASC)*, Sep. 2016, pp. 315–318.

[12] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 1–42, Feb. 2012.

[13] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2017, pp. 239–248.

[14] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2016, pp. 183–194.

[15] B. Jung, T. Kim, and E. G. Im, "Malware classification using byte sequence information," in *Proc. Conf. Res. Adapt. Convergent Syst. (RACS)*, 2018, pp. 143–148.

[16] H. Xue, S. Sun, G. Venkataramani, and T. Lan, "Machine learning-based analysis of program binaries: A comprehensive study," *IEEE Access*, vol. 7, pp. 65889–65912, 2019.

[17] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *J. Comput. Virol. Hacking Techn.*, vol. 14, no. 1, pp. 1–20, Sep. 2016.

[18] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Berlin, Germany: Springer, Jul. 2013, pp. 41–61,

[19] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "MutantX-S: Scalable malware clustering based on static features," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 187–198.

[20] B. G. Ryder, "Constructing the call graph of a program," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[21] F. E. Allen, "Control flow analysis," *ACM SIGPLAN Notices*, vol. 5, no. 7, pp. 1–19, Jul. 1970.

[22] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *J. Comput. Virol.*, vol. 7, no. 4, pp. 233–245, Feb. 2011.

[23] O. Kostakis, J. Kinable, H. Mahmoudi, and K. Mustonen, "Improved call graph comparison using simulated annealing," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2011, pp. 1516–1523.

[24] L. Xu and E. Oja, "Improved simulated annealing, Boltzmann machine, and attributed graph matching," in *Proc. Eur. Assoc. Signal Process. Workshop*. Berlin, Germany: Springer, Feb. 1990, pp. 151–160.

[25] X. Hu, T.-C. Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, 2009, pp. 611–620.

[26] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, and N. Zheng, "A similarity metric method of obfuscated malware using function-call graph," *J. Comput. Virol. Hacking Techn.*, vol. 9, no. 1, pp. 35–47, Jan. 2013.

[27] T. Dullien and R. Rolles, "Graph-based comparison of executable objects (English version)," in *Proc. SSTIC*, 2005, vol. 5, no. 1, p. 3.

[28] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2013, pp. 1357–1365.

[29] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[30] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complex. SEQUENCES*, Jun. 1997, pp. 21–29.

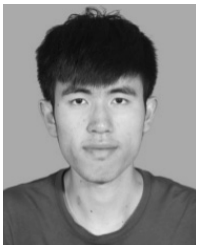
[31] X. Gao, C. Shan, C. Hu, Z. Niu, and Z. Liu, "An adaptive ensemble machine learning model for intrusion detection," *IEEE Access*, vol. 7, pp. 82512–82521, 2019.

[32] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, 2014.

[33] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv:1804.04637*. [Online]. Available: http://arxiv.org/abs/1804.04637

YIPIN ZHANG is currently pursuing the master's degree with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University. Her research interests include network security and machine learning.

XIAOLIN CHANG (Member, IEEE) is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include edge/cloud computing, network security, and security and privacy in machine learning.



YUZHOU LIN is currently pursuing the master's degree with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University. His research interests include malware detection and machine learning.

JELENA MIŠIĆ (Fellow, IEEE) is currently a Professor of computer science with Ryerson University, Toronto, ON, Canada. She has published four books, over 125 articles in archival journals, and close to 190 articles at international conferences in the areas of computer networks and security. She is a member of ACM. She serves on the Editorial Board of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE INTERNET OF THINGS JOURNAL, the *IEEE Network*, *Computer Networks*, and *Ad hoc Networks*.

VOJISLAV B. MIŠIĆ (Senior Member, IEEE) is currently a Professor of computer science with Ryerson University, Toronto, ON, Canada. His research interests include performance evaluation of wireless networks and systems and software engineering. He is a member of ACM. He serves on the Editorial Boards of the IEEE TRANSACTIONS ON CLOUD COMPUTING, *Ad Hoc Networks*, *Peer-to-Peer Networks and Applications*, and the *International Journal of Parallel, Emergent and Distributed Systems*.