IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 24, 2019, accepted July 30, 2019, date of publication August 8, 2019, date of current version August 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2934012

SMASH: A Malware Detection Method Based on Multi-Feature Ensemble Learning

YUSHENG DAI[1], (Student Member, IEEE), HUI LI[1], YEKUI QIAN[2], RUIPENG YANG[2], AND MIN ZHENG[3]

[1]School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710129, China
[2]National Digital Switching System Engineering and Technological R&D Center (NDSC), Zhengzhou 450001, China
[3]Henan Institute of Information Security Company Ltd., Xuchang 461000, China

Corresponding authors: Yusheng Dai (daiyusheng@mail.nwpu.edu.cn) and Yekui Qian (qyk1129@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61571364, and in part by the Innovation Foundation for Doctoral Dissertation of Northwestern Polytechnical University under Grant CX201952.

ABSTRACT With the increasing variants of malware, it is of great significance to detect malware and ensure system security effectively. The existing malware dynamic detection methods are vulnerable to evasion attacks. For this situation, we propose a malware dynamic detection method based on mufti-feature ensemble learning. Firstly, the method adopts the combination of software features such as API call sequence with high detection precision and low-level hardware features such as resistance to evasion the memory dump grayscale and hardware performance counters. Secondly, we improve each feature based on the original research. We select a more advanced classifier model to improve the detection precision of a single feature. Finally, an ensemble learning algorithm composed of multiple classification algorithms detects malware, the multi-features can describe malware behavior from multi-dimensions to improve detection performance. We use a large number of malware sample dataset to experiment, and the results show that our detection method can obtain good detection precision rate, and is better than other recently proposed dynamic detection methods in anti-evasion performance.

INDEX TERMS Anti-evasion, dynamic detection, hardware features, memory dump.

I. INTRODUCTION

With the continuous development of information technology, cybercrime is a serious threat to the economic, military and other important areas of various countries. Malware is one of the important factors that undermine Internet security. Malware has grown rapidly in both quantity and categories compared with foretime. New malware, especially Advanced Persistent Threat (APT), is more and more difficult to be detected by current defection technologies.

The main ways of detecting malware include dynamic detection and static detection [1]. Dynamic detection can effectively monitor the running behavior and state of malware, and has received extensive attention from researchers. However, dynamic detection is vulnerable to evasion attacks [2]. According to the different detection system models, malware authors can set different countermeasures to evade detection according to the detection model. At present, there are a large number of malware dynamic detection studies, and a good detection rate can be obtained [3]–[11], but malware with evasion behavior cannot be effectively handled. Commonly used dynamic detection evasion methods use code reuse technical against detectors [12]–[14] or use mimicry attack to evade detectors [15], [16]. Smutz and Stavrou [17] proposed the method of using mutual agreement analysis combined with ensemble learning for evasion malware detection. On the PC platform, using API call sequences is more effective than behavior-based dynamic detection methods for characterizing malware [18]. However the software itself has the same defect density [19], the method of using software features for detection is vulnerable to evasion attacks. In addition, the detector of dynamic detection itself can obtain enough evasion details through the black box test [17]. Due to the shortcomings of dynamic detection that are easily evasion, some recent studies focus on malware detection based on hardware features to avoid the lack of software features. Demme *et al.* [20] demonstrated that malware can be effectively detected by extracting hardware performance counter (HPC) information in dynamic detection. Based on research by Demme et al.,

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Yuan Chen.

Khasawneh *et al.* [21], [22] developed an integration method featuring a variety of HPC information for malware detection. However, the above-mentioned that using only hardware feature method cannot fully describe the characteristics of malware, resulting in a low detection rate, and is vulnerable to evasion attacks aiming at performance counters as classification features.

Aiming at the defects of dynamic detection and the shortcomings of insufficient hardware detection rate, in this paper, we proposes an ensemble learning method (SMASH) that combines software and hardware features, extracts API call sequence of malware, hardware performance counters and memory dump [23] as detector features, and builds different types of feature vectors. Wherein, the software features make up for the lack of hardware feature detection precision, and the hardware feature compensation software features are vulnerable evasion. An existing neural network with excellent detection performance is used as a detector, so that each feature corresponds to a specific detector for malware classification, and all the detection results are voted to determine the maliciousness of the tested sample.

This paper mainly contributes 3 points as follows: (1)

1) We propose a method that combines software and hardware features, API call sequences as software features, HPC and memory dump grayscale images as hardware features. This method describe malware behavior from multi-dimensions, and can effectively combat the evasion of malware.
2) We use advanced neural networks to improve the detection performance of each feature separately. Combine the results of each feature detection to determine the weight of the ensemble detector through experiments, and determine the test results by voting in the end.
3) The effectiveness of this method is verified by using actual malware and malware samples with evasion performance for experimental evaluation.

The remaining sections of this paper are structured as follow: Section 2 summarizes the related work; Section 3 describes the SMASH method in detail, namely the feature extraction method and algorithm model; Section 4 evaluates our method from the detection performance and the anti-evasion performance; Section V concludes the full text and proposes further work.

II. LITERATURE REVIEW

Malware detection is mainly divided into static detection and dynamic detection [1]. The static detection method generally adopts the disassembly mode, which can understand the program from the grammar and semantics. It has the advantages of simple and efficient. Arp *et al.* [24] used multiple vector combinations based on semantics and syntax to detect malware for Android and document files. Kapoor and Dhavale [25] disassemble executable files and generate control flow graphs, then extract features from n-grams and extract features grading, and finally used information gain to feature dimensionality reduction and using machine learning algorithms to detect malware. Nguyen *et al.* [26] proposed an enhanced form of control flow graph, and converted the control flow graph into an image, then used deep learning to detect malware. However, these methods are subject to defects in static detection. Nataraj *et al.* [27] and Nataraj and Manjunath [28] first proposed a method for converting a malware file into a grayscale image, and used the texture feature of the image to classify the malware, and visual method is used to compensate for the static test vulnerable to confuse and encryption methods, but still cannot solve the problem of padding code.

Dynamic detection technology can effectively overcome the shortcomings of static analysis technology, and it attracts extensive attention because it can monitor the behavior of samples while the program is running [29]. The studies on malware dynamic detection have attracted extensive attention [7]–[10], and all have obtained good detection accuracy. The common detection features in dynamic detection include behavior-based malware dynamic detection [3]–[6], and dynamic detection studies using API call sequences [11], [30], [31], all of which use software features to detect malware. Since the API call sequence is easy to be extracted and has relatively high detection accuracy, the malicious behavior of the tested sample can be fully described, but malware with evasion behavior in these work cannot be effectively detected. In the study of malware detection, as the malware itself has functionality, it is relatively difficult to provide suitable countermeasures. Such confrontation is called analog attack [32]. Currently, research on mimicry attack of malware includes [16], [33], [34], and these attacks use PDF as an attack payload to evade the detector.

Recently, there are quite a few research methods based on hardware features. As there is no essential difference between the software and general software regarding the vulnerability, they are vulnerable to different degrees of confrontation. Demme *et al.* [20] found that collecting performance counter information of malware at runtime can effectively distinguish malware through offline analysis and verify the effectiveness of using hardware performance counter. By using HPC and its related events, Tang *et al.* [19] demonstrated that using off-line unsupervised analysis method can also effectively analyze new types of malware or family evolution. Ozsoy *et al.* [35], [36] and Khasawneh *et al.* [21], [22] adopted the two-level detection method, they use a variety of low-level hardware features for pre-detection, raise the weight of suspicious files, and recheck with software detector. However, the malware behavior cannot be described perfectly by using the hardware detection method, because the features are too single. The study by Zhou *et al.* [37] shows that HPC cannot effectively correspond to the upper API in describing malicious behavior, resulting in a decrease in accuracy. The study by Das et al. IEEEexample:das2019sok shows that HPC may also be exploited by an attacker to cause an evasion of detection. Dai *et al.* [23] used HOG extracted from memory dump grayscale image as a feature to verify the validity of the feature classification detection, which could effectively describe the malware to some extent.

Most of the above methods focus on detection accuracy, while there is less concern about the evasion of malware. Smutz and Stavrou, [17] proposed the use of mutual agreement analysis combined with multiple classifier collection, and hope it can combat the evasion from the detector, but using only software features for detecting malicious software with evasion behavior still results in undetermined detection. Khasawneh *et al.* [38] designed a hardware detector, used performance counters as features and re-training resilient method to resist the evasion attack. However, using only hardware features still has a problem of a low detection rate.

III. SMASH METHOD

The SMASH method proposed in this paper mainly has three steps. The first step is to use dynamic sandbox to extract the three features of malware, namely API call sequence, memory dump grayscale image and performance counter count; the second step is to train and predict each feature in a specialized classifier; the final step is to perform integrated computing of the test results to determine the maliciousness of the input samples. The experimental process is shown in Fig. 1.

A. API SEQUENCE EXTRACTION

The software features used in the SMASH method of this paper are based on the API call sequence method, and the API call sequence can demonstrate the optimal detection performance on the PC platform [18]. By monitoring all the ring0 level API functions executed by the malware in the dynamic sandbox, the sequence of function execution is extracted as a feature for malware classification and detection work. In this paper, the API call sequence is used as a feature, word2vec is used to convert the malware API call sequence into a vector, and a multi-layer BiGRU network is used to detect malware.

In this paper, the cuckoo sandbox [39] is used to extract all the features of malware samples, including API call sequence, memory dump files, and HPCs. Here we use the report package of the cuckoo sandbox to get the API call sequence of malware samples. Due to the different call order of different kinds of application APIs, the calling combination of some functions will appear more frequently in malware. In order to convert the obtained API sequence into a feature vector that can be recognized by the algorithm, the word2vec method is used in processing API sequence in this paper.

The corpus of CSDMC 2010 API [30] is used in our word2vec method. Combined with the data sample API sequence used in the experiment, a hierarchical softmax-based CBOW model is used. The extracted API sequence form a text sequence $f = \{w_1, w_2, w_3, \ldots, w_n\}$ according to the call sequence, where w_n represents the word vector of the corresponding function, and set maximum distance between the current and predicted function is $c = 2$. Therefore, the front and back for each word in the text sequence f can be calculated:

$$context(w)_n = \frac{1}{2c}\sum_{i=i}^{2c} w_i \tag{1}$$

In $v(context(w)_n) \in \mathbb{R}^m$, w_i is the i-th neighbor of w, and m is the word vector length. Then, through the gradient rise, after setting 20 iterations, a set of vectors $V = \{label, v_1, v_2, v_3, \ldots, v_n\}$ related to the sample API sequence is finally output in combination with the label, and each word vector value is mapped to the file accordingly. Other APIs are filled with 0 to form fixed length vector V_S.

B. MEMORY DUMP GRAYSCALE IMAGE EXTRACTION

The first hardware feature used by the SMASH method is the grayscale image converted by the memory dump file. Memory dump can store more information, so it can describe the behavior of malware more comprehensively, and can effectively detect malware with evasion behavior to a certain extent (Dai *et al.* [23]). Memory dump refers to the volatile data extracted from the computer physical memory and swap page, and it is the important data used in memory forensics this computer branch science. Memory dump usually has full memory dump, core dump, process dumps, and so on. Here we extract and analyze the sample process memory dump file to determine whether this process is malicious.

The memory dump of a process typically contains the dynamic link libraries (DLLs), environment variables, process heaps, thread stacks, data segments, and text segments required by the process. In order to extract the transferred files, we extract the latest memory dump at a fixed time in the cuckoo sandbox and extract it into the sandbox's Host system. In order to make the detector correctly recognize the read dump data, we map the dump file to fixed-size grayscale images.

A given dump file is read from the beginning of the file with 8 bit unsigned int as step size and converted into a one-dimensional array in whole. The computer's single-channel grayscale image has a maximum gray level of 256. Each byte is read sequentially from a one-dimensional array and converted to single-channel PNG format file according to fixed line width.

Since the memory size of dump file generated by each sample is inconsistent, in order to get grayscale image with consistent size and the image is not significantly distorted, the bicubic interpolation is used to compress the grayscale image. During the processing of grayscale image, cubic interpolation is performed on the 16 pixels in the adjacent 4 × 4 area, taking into account not only the grayscale effect of 4 directly adjacent points, but also the effect of rate of change for gray value between adjacent points. The bicubic interpolation of 4 × 4 pixels is expressed as follows:

$$p(x, y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_i j x^i y^j \tag{2}$$

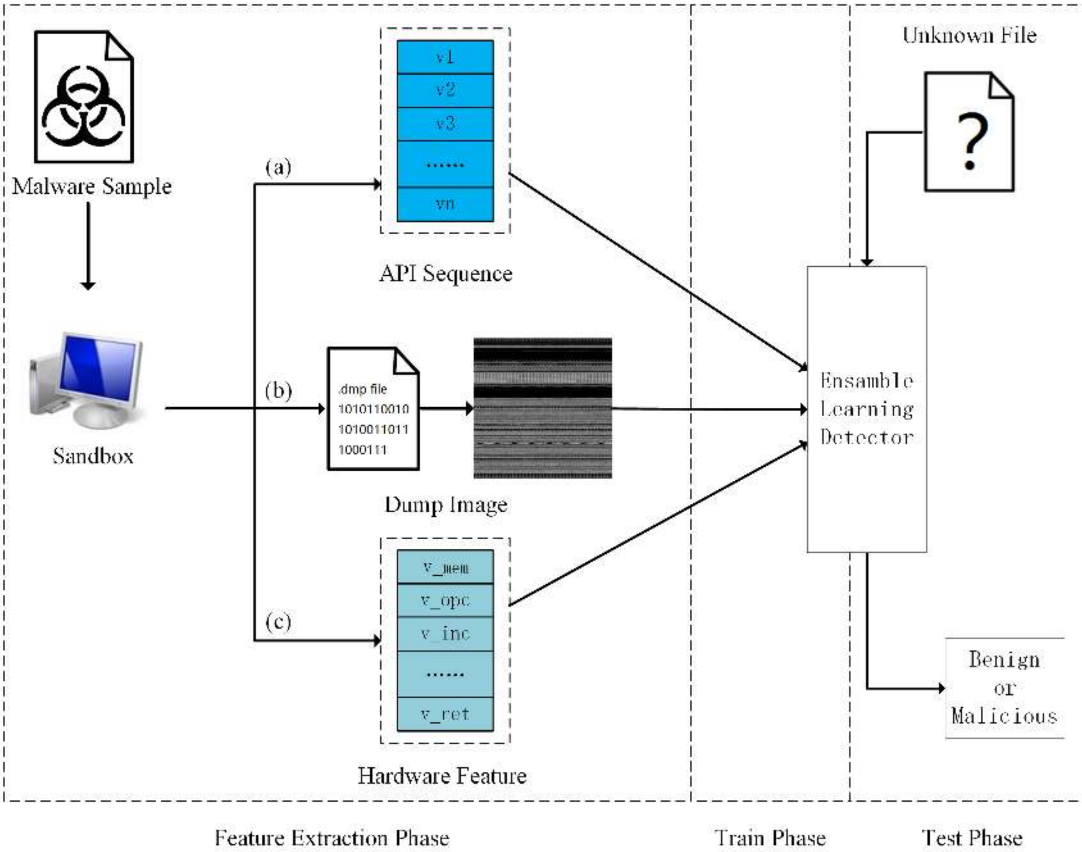

FIGURE 1. SMASH method architecture.

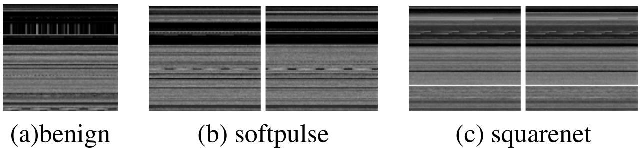

FIGURE 2. Example of sample dump grayscale image.

where (x, y) is the pixel point to be interpolated in grayscale image, and a_ij is the 4 × 4 neighborhood point around pixel point. The tagged feature data $V_G = \{label, A_{224\times224}\}$ can be obtained by outputting the image to a grayscale image with equal side length. Fig. 2 shows the example of sample dump grayscale image extracted by different families.

C. LOW-LEVEL FEATURE EXTRACTION

Hardware low-level features include architectural features and micro-architectural features. This work refers to the work of Demme *et al.* [20] and Ozsoy *et al.* [35], [36], using the hardware performance counter of CPU as the features used in our SMASH method.

We programmatically implant a soft probe on the guest side of cuckoo to read the CPU's Model Specific Registers (MSR). We set virtualized performance counters in virtual machine, and MSR can record the current CPU performance status. Three fixed performance counters (instruction retired, unhalted core cycle, etc.), and four optional event performance counters. The capturing runtime behavior of performance counters using hardware can be used to identify malicious software and to ensure that minor changes in malicious software during monitoring process will not significantly interfere with the detection process.

According to the characteristics of different architectural events [19] and with reference to the conclusions of [20], [35], [36], combined with the data used in our experiments, we only select three counters that can reflect the optimal performance as hardware features of HPC. Total performance status of CPU is denoted as V_P, Instructions Retired is denoted as V_IR, and Unhalted Core Cycle is denoted as V_UC. The sampling frequency is about 3,000k clock cycles under 3.2 GHz CPU, which is a relatively reasonable sampling frequency under Windows system. Excessive sampling frequency will cause higher expenditure and further increase the sampling noise.

D. CLASSIFICATION MODEL

1) BIGRU NEURAL NETWORK

In this work, the extracted API call sequence feature vector and low-level hardware feature vector are correlated in time. Therefore, the multi-layer BiGRU network is selected

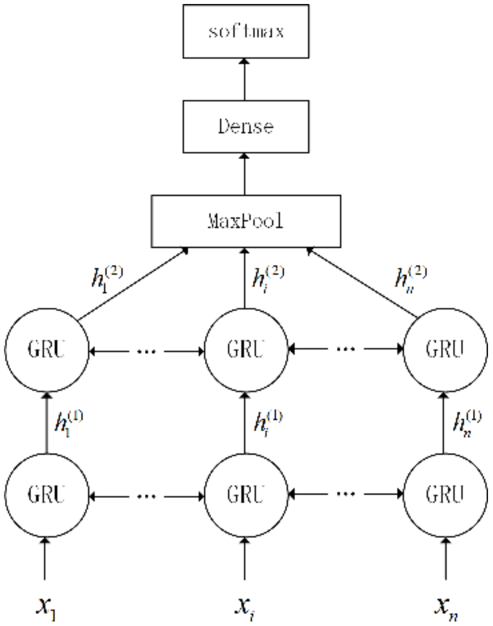

FIGURE 3. Two-layer BiGRU network structure.

to detect the API sequence features and low-level hardware features. GRU is a variant of recurrent neural network, and LSTM neural network is similar to it. Because it is applicable to the characteristics of detecting sequential characteristic and has fewer parameters than LSTM, it is easy for convergence.

Fig. 3 shows the overview of network structure for training feature data in this work. API sequence or hardware performance counter feature vector $X = \{x_1, \ldots, x_i, \ldots, x_n\}$ is input in model, and x_i is the input of a certain time point. In order to calculate a GRU cell hidden layer output h_i, first it is required to calculate the updated gate z_i and reset gate r_i in the cell, and user r_i to calculate the $\tilde{h}_i$ of candidate memory cell, then finally obtain the h_i output of current cell. The calculation process of single GRU cell is as follows:

$$z_i = \sigma(W_z x_i + U_z h_{i-1}) \tag{3}$$

$$r_i = \sigma(W_r x_i + U_r h_{i-1}) \tag{4}$$

$$\tilde{h}_i = tanh(Wx_i + U(r_i * h_{i-1})) \tag{5}$$

$$h_i = (1 - z_i) * h_{i-1} + z_i * \tilde{h}_i \tag{6}$$

where h_{i-1} is the cell status information of the previous moment, and W, U is weight matrix. In our network, a GRU cell represents the API in a feature vector, or the value of a hardware feature at the current time. Bilaterally used to calculate the relationship between the features "past" and "future" to increase the number of GRU layers, so as to increase the learning ability of the whole network, and the data is more and more abstract. In combination with the use of two-layer BiGRU in above content, the state of upper layer is used as input to each GRU cell in the next layer.

The maximum pooling layer is used to process information of hidden layer. Probability output is achieved from the data obtained after pooling passing through the fully connected layer and then through softmax function. The overhead of the BiGRU network used in the method is measured by the parameter quantity and the computing power, wherein the parameter quantity is about 21.56M, and the computing power is counted using the number of floating-point operations, and the computing power is about 4.32GFlops.

2) VGG19

In section 4.2, we have extracted the grayscale image mapped by malware memory dump. This paper uses convolutional neural network to classify the grayscale image of memory dump. Convolutional Neural Network is a multi-layer neural network structure that is composed of input layer, convolution layer, pooling layer, fully connected layer and output layer. The input layer inputs the grayscale image of memory dump; The pooling layer extracts the features of grayscale image; The pooling layer uses the local correlation principle of image to reduce the volume of data to be processed; The output layer maps the features to final predicted results. Next, the VGG19 network used in our work will be described.

a: CONVOLUTION LAYER

the output of the upper layer can be the input of the current convolution layer. In order to improve the expression ability of network model, the nonlinear activation function is introduced. The forward propagation of convolution layer k can be expressed as: $C^k = \sigma(C^{k-1} * W^k + b^k)$. Where W is the high-dimensional tensor of convolution kernel, b is bias, and Relu is used as activation function σ.

b: POOLING LAYER

Pooling function uses the overall statistical characteristics of the adjacent output at a position to replace the network output at the position, which can carry out dimension reduction on the output of previous convolution layer and remain most outputs unchanged in the meantime. Set pooling function as the maximum sampling, and use the sliding window of 2×2 with a step of 2.

Fully connected layer and output layer: All the two-dimensional feature maps outputted by the convolution layer are spliced into one-dimensional feature vector as the input of fully connected layer. The fully connected layer obtains the output $h^k = \sigma(W^k h^{k-1} + b^k)$ of Layer k by the weighted sum of input and the activation function. Here, the activation function still uses Relu, and W is the weight matrix of the current layer, while b is bias. The final output layer, based on the output of the previous fully connected layer, uses softmax function regression to convert the results of neural network forward propagation into probability distribution.

The convolutional neural network used for grayscale images in this work uses a single channel image as input. The network model is shown in Table 1, wherein, stride is the step length of convolution kernel movement, pad is padding, true

TABLE 1. VGG19 model.

Name	Kernel size	Stride	Pad	Function
conv1_1	$3 \times 3 \times 64$	1	true	Relu
conv1_2	$3 \times 3 \times 64$	1	true	Relu
pool1	2×2	2	false	MaxPool
conv2_1	$3 \times 3 \times 128$	1	true	Relu
conv2_2	$3 \times 3 \times 128$	1	true	Relu
pool2	2×2	2	false	MaxPool
conv3_1	$3 \times 3 \times 256$	1	true	Relu
conv3_2	$3 \times 3 \times 256$	1	true	Relu
conv3_3	$3 \times 3 \times 256$	1	true	Relu
conv3_4	$3 \times 3 \times 256$	1	true	Relu
pool3	2×2	2	false	MaxPool
conv4_1	$3 \times 3 \times 512$	1	true	Relu
conv4_2	$3 \times 3 \times 512$	1	true	Relu
conv4_3	$3 \times 3 \times 512$	1	true	Relu
conv4_4	$3 \times 3 \times 512$	1	true	Relu
pool4	2×2	2	false	MaxPool
conv5_1	$3 \times 3 \times 512$	1	true	Relu
conv5_2	$3 \times 3 \times 512$	1	true	Relu
conv5_3	$3 \times 3 \times 512$	1	true	Relu
conv5_4	$3 \times 3 \times 512$	1	true	Relu
pool5	2×2	2	false	MaxPool
fc1	4096			Relu
fc2	512			Relu
predict	2			softmax

is using 0 for padding, false is no padding; Function is the function used by the current layer.

The overhead of our VGG19 network model is measured by parameter quantity and the computing power, where the parameter quantity is about 139.56M, and the computing power is about 17.31GFlops.

3) ALGORITHM INTEGRATION

Since a single detector uses a single algorithm and is easy to be attacked by reverse analysis mechanism to some extent, we combine all the detectors to improve the overall complexity and detection performance of the detector. We apply multiple features to different detectors and combine the decisions made by all detectors into the final decision, as shown in the training and detection phases of Fig. 1.

The method of integrating multiple algorithms is called ensemble learning, which is suitable for integrating the decisions made by multiple individual detectors into a complete decision, so in the work, all the results of detector are subject to integrated voting method to detect if an unknown sample is malicious. We use a weighted voting method for any sample x, and our detection results for t detectors are respectively $\{h_1(x), h_2(x), \ldots, h_{t-1}(x), h_t(x)\}$. We set each feature, taking the performance as the original weight, and then fine-tuning according to the experimental results to choose the optimal weight. The final predicting outcomes can be obtained according to different weights of detectors. The expression is shown in the following formula, where ω_i is the weight of the i-th detector.

$$H(x) = \sum_{i=1}^{t} \omega_i h_i(x) \tag{7}$$

TABLE 2. Dataset category.

Sample category	Category No.	Quantity
Backdoor	0	4652
Flooder	1	527
Worm	2	3821
Exploit	3	541
Trojan	4	2047
Adware	5	3226
Constructor	6	602
Hacktool	7	655
Other Malware	8	662
Benignware	9	1119

IV. EXPERIMENTAL EVALUATION

A. EXPERIMENTAL ENVIRONMENT AND DATASET

The computer environment of sandbox environment running is using CPU of Intel(R) Core(TM) i5-6500 @3.20GHz; using 8GB DDR3 memory The Host machine of cuckoo sandbox is installed under Ubuntu16.04 system environment with Guest machine using Windows7 32-bit operating system, 2GB memory.

SMASH method runs on a graphic workstation with hardware environment using CPU of Intel Core (TM) i7-6800K processor chip; 32GB dual channel DDR4 memory; The graphics card is Nivdia 1080Ti with 11GB of video memory.

The malware dataset used in this paper comes from OpenMalware [40] and the use of malware is authorized with a total of about 27k malware samples and an acquisition range of 2013-2015; The evasion sample data is about 100, collected in 2016-2017 and written by ourself. All samples can be classified into 214 families by VirusTotal [41] according to their families. The classification according to malware functions and the quantity information used in the experiment can be found in Table 2.

B. EVALUATION CRITERION

This experiment uses accuracy ($Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$), precision ($Precision = \frac{TP}{TP+FP}$), recall ($Recall = \frac{TP}{TP+FN}$) and F1-Score ($F1 - Score = \frac{(2*Precision*Recall)}{(Precision+Recall)}$) these four indicators to measure the effectiveness of classification using the method in this paper combined with machine learning, wherein, the TP, FP, TN and FN respectively represent true positive, false positive, true negative and false negative cases. Accuracy refers to the quantity of data which is classified correctly by the classifier in the classification process, while recall refers to the quantity of data that can be found in all correct data. As the accuracy and recall in classification process is a pair of contradictory measures, the use of F1-Score can effectively balance the accuracy and recall, and the closer to 1 of F1-Score numerical value means better classifier performance.

Meanwhile, we use the ROC (Receiver Operating Characteristic) curve and AUC (Area Under Curve) area to reveal the performance of each feature on different detectors, integrate the overall detection performance of detectors, and evaluate the performance of detectors on malware dataset with evasion features.

TABLE 3. Indicator performance of multiple features.

Features	Precision	Recall	F1-Score
API Sequence	96.5%	97.6%	97.0%
Gray Image	92.9%	91.5%	92.0%
General Performance	92.0%	90.1%	91.0%
Instructions Retire	90.9%	87.5%	89.3%
Unhalted Cycle	88.6%	89.5%	89.0%

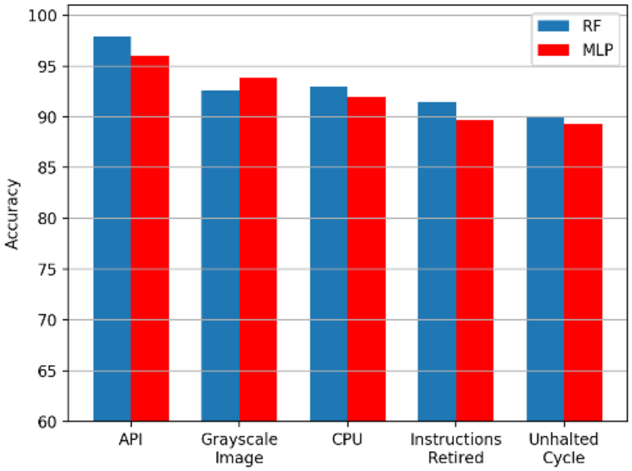

FIGURE 4. Multiple features use the accuracy of different detectors.

TABLE 4. Comparison of single classifier and SMASH results.

Classifier	Accuracy	Precision	F1-Score
SVM	94.3%	94.5%	94.6%
RF	95.1%	95.6%	96.1%
Bi-GRU(API)	97.4%	98.0%	97.3%
Bi-GRU(HPC)	90.4%	91.0%	91.0%
VGG19	93.5%	93.1%	93.6%
SMASH	96.9%	97.8%	98.1%

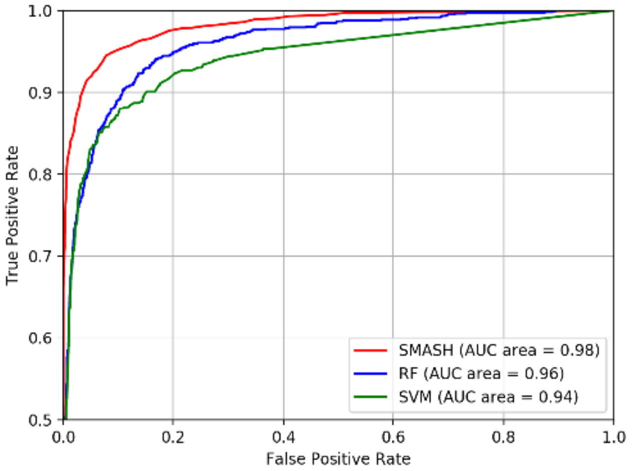

FIGURE 5. Performance comparison of three classifiers.

C. SMASH METHOD EVALUATION

1) FEATURE EVALUATION

In the experiment, we take 80% of the samples in all available malware as a training set and take 20% of the samples as a test set. In SMASH experiment, we use an integrated detector (Random Forest, RF) and a universal detector (multilayer perceptron, MLP) to compare the same features so as to verify the performance of each feature in malware detection. The different features shown in Fig. 4 use the accuracy histograms of the two detectors. Table 3 lists the best detection rate, recall, and F1-Score for each feature.

It can be seen from Fig. 4 that no matter which detector is used, the accuracy of using API sequence as a feature to detect malware is better than the hardware feature. Regardless of the classification in accordance with family or detection, API sequence feature can show good performance. In terms of hardware features, the detection rate combining the memory dump grayscale image and the overall performance counter of CPU shown in Table 3 is relatively close, and the accuracy is high, whereas the detection rates of other performance counter features are relatively low. On the whole, the features we selected as input data of detector should have a high detection rate.

2) CLASSIFIER EVALUATION

We use API call sequence and low-level hardware performance counters as features, and use BiGRU neural network as a detector; use memory dump grayscale image extracted by malware as a feature, and use VGG19 convolutional neural network as a detector. Meanwhile, we select two widely used classifier, support vector machine (SVM) and random forest (RF) to train and test the features extracted in the experiment using integrated approach respectively. The experiment verifies the detection rate of each classifier and compares with the detection rate of SMASH method. The results are shown in Table 4:

In this group of experiments, we use a BiGRU neural network to detect malware API sequences and use VGG19 to detect malware memory dump grayscale image. It can be seen from the accuracy and F1-Score that the detection rate is higher compared with the use of random forest and multi-layer perceptron in Table 3. As can be seen from Table 4, using BiGRU in combination with three HPC features are superior to the mean value of single performance counter's optimal performance in Table 3. In terms of integrated detector, we use support vector machine and random forest. Each individual detector corresponds to a feature, and it is compared with SMASH method based on the result of weighted sum. SMASH method can reach 96.9% and 98.1% on the accuracy and F1-Score.

Fig. 5 shows the ROC curves of three integrated detectors. The x-axis of ROC curve image is the False Positive Rate and the y-axis is the True Positive Rate. It can be seen from the figure that, the AUC value of SMASH method is about 0.98, which is about 4% higher compared with that of support vector machine method and random forest method using integration.

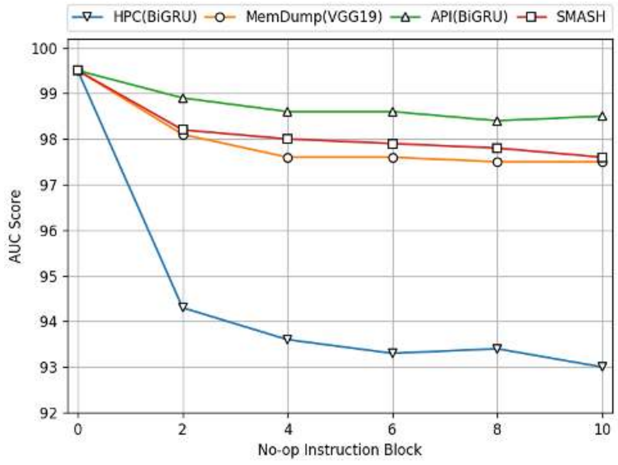

FIGURE 6. Detection performance of Non-op instruction block injection.

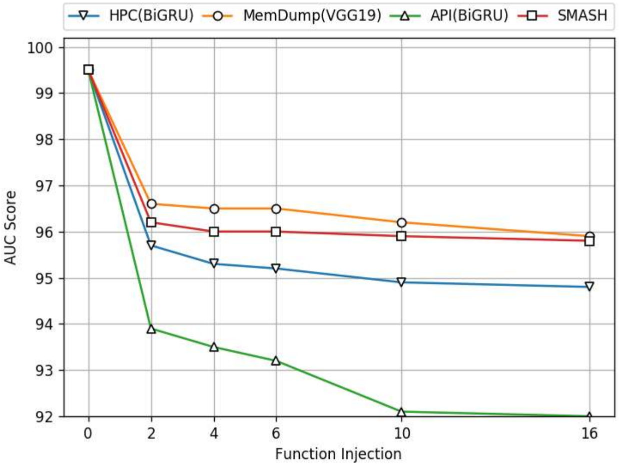

FIGURE 7. Detection performance of benign function injection.

D. ANTI-EVASION PERFORMANCE EVALUATION

The same family of malware constantly generates new samples, which is an evasion process itself [17], and meanwhile, as the detector method ages, for the newly generated malware, the detection rate will be greatly affected [42]. Usually the author of malware will use a special or general attack strategy to evade or bypass the defense system of targeted victim according to the defense strategy of the target attacked. Our experiments imitate the evasion behavior and assume that the attacker knows nothing about our detector, i.e. the black box testing.

1) RANDOMIZE INSTRUCTION INJECTION

In the experiment, we set malware evasion type to randomly injected instruction malware in a way of inserting invalid instructions to avoid detection. In a way of inserting invalid instruction block, we randomly insert non-operational instruction block into the code, and the number of random insertions is gradually increased. AUC area is used as the annotation to measure the performance of each feature, as shown in Fig. 6.

As can be seen from the figure, HPC is greatly affected by inserting invalid instruction block, but API sequence and grayscale image methods are less affected. Due to the insertion of invalid instruction, the count of performance counter is distorted during the same sampling period and detector performance is degraded. Whereas the API operation sequence and grayscale image resist evasion attacks on a relatively large scale.

2) BENIGN INSTRUCTION INJECTION

Injecting benign instruction to reduce the detector's detection rate of malware is another evasion method. In the experiment, we set the function that can reflect benign behavior to be injected into malware, and the number of function injections is gradually increased, as shown in Fig.7.

TABLE 5. Comparison with other methods.

Other methods	Features	Accuracy	F1-Score
Smutz et al.	API	91.9%	92.5%
EnsambleHMD	HPC	88.5%	90.8%
SMASH	Multi-Feature	94.9%	94.1%

It can be seen from the figure that, the detection performance of API sequence will be seriously affected by injecting benign function, while HPC and memory dump grayscale image are relatively less affected. It is a challenge in dynamic detection that benign API sequence is introduced to affect API detector performance. We use memory dump grayscale image as a feature, which is able to effectively defend against such attacks. Because API occupies text segments in memory, and this part is only a part of process memory, there are still other related data that can be used as a feature to express sample maliciousness.

Therefore, it can be seen from the experimental results of random injection no-op blocks (Fig. 6) and injection benign function (Fig. 7), our SMASH method can effectively against two evasion attacks, in which memory dump provides higher anti-evasion performance contribution. Since the memory dump basically contains no invalid instructions, and the instructions in the memory are stored in the text segment, it is only part of the memory and has little effect on the process memory. However, memory dump has a large gap with API call sequence in detection precision, so using memory dump image can alleviate the evasion attack of malware to a certain extent.

3) COMPARISON WITH OTHER RESEARCH

In order to verify the effectiveness of our method, we compare the research of Smutz and Stavrou [17] with the Ensamble HMD of Khasawneh *et al.* [22] using our dataset, and the optimum results are taken. See Table 5.

It should be pointed out that Ensamble HMD is a hardware-based detector, and our recurring experiments use only the malware features used in this research and the offline universal detector (neural network) method in this research. Since it is soft probe we use, and some noise may also be introduced when acquiring HPC, the difference between the two experiment settings is not counted therein. As can be seen from the table, the multi-featured integrated detector used in our SMASH method has the highest accuracy.

V. CONCLUSION

The existing dynamic detection method based on software features cannot effectively deal with the problem of malware evasion, and the detection method based on hardware features also suffers from imitation attacks against features and detectors. In response to this problem, this paper proposes software and hardware features combining API feature sequence, memory dump grayscale image, hardware performance counter, etc., and uses the corresponding neural network to detect malware for each feature. By improving the detector model for each feature and using the ensemble learning method, the detection precision of malware can be optimized, and the detection accuracy can be as high as 97.8%. The effectiveness of the method proposed in this paper is verified by using different types of malware and the malware containing evasion behavior. In the experiment, the detection precision decreased by no more than 3%, and the performance against evasion attacks is slightly better than other recent studies. In future work, we will research the types of malware that are currently difficult to detect (COOP, Powershell, etc.), which are considered to be highly threatening in current research.

ACKNOWLEDGMENT

The authors would like to thank the editor and the anonymous referees for their constructive comments.

REFERENCES

[1] N. Idika and A. P. Mathur, "A survey of malware detection techniques," Purdue Univ. West Lafayette, IN, USA, Tech. Rep., 2007, vol. 48.

[2] A. Bulazel and B. Yener, "A survey on automated dynamic malware analysis evasion and counter-evasion: PC, mobile, and Web," in *Proc. 1st Reversing Offensive-Oriented Trends Symp.*, 2017, p. 2.

[3] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.

[4] Y. Ding, X. Xia, S. Chen, and Y. Li, "A malware detection method based on family behavior graph," *Comput. Secur.*, vol. 73, pp. 73–86, Mar. 2018.

[5] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jun. 2016, pp. 577–582.

[6] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Feb. 2016, pp. 1–5.

[7] B. Alsulami, A. Srinivasan, H. Dong, and S. Mancoridis, "Lightweight behavioral malware detection for windows platforms," in *Proc. 12th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2017, pp. 75–81.

[8] M. Yousefi-Azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics: A malware detection scheme," *IEEE Access*, vol. 6, pp. 49418–49431, 2018.

[9] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 2482–2486.

[10] B. J. Kwon, V. Srinivas, A. Deshpande, and T. Dumitra , "Catching worms, trojan horses and PUPs: Unsupervised detection of silent delivery campaigns," 2016, *arXiv:1611.02787*. [Online]. Available: https://arxiv.org/abs/1611.02787

[11] A. G. Kakisim, M. Nar, N. Carkaci, and I. Sogukpinar, "Analysis and evaluation of dynamic feature-based malware detection methods," in *Proc. Int. Conf. Secur. Inf. Technol. Commun.* Springer, 2018, pp. 247–258.

[12] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy, "Return-oriented programming without returns," in *Proc. 17th Conf. Comput. Commun. Secur.*, 2010, pp. 559–572.

[13] E. Bosman and H. Bos, "Framing signals—A return to portable shellcode," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 243–258.

[14] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A.-R. Sadeghi, and T. Holz, "Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 745–762.

[15] D. Bruschi, L. Cavallaro, and A. Lanzi, "An efficient technique for preventing mimicry and impossible paths execution attacks," in *Proc. IEEE Int. Perform. Comput. Commun. Conf.*, Apr. 2007, pp. 418–425.

[16] D. Maiorca, D. Ariu, I. Corona, and G. Giacinto, "A structural and content-based approach for a precise and robust detection of malicious PDF files," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, Feb. 2015, pp. 27–36.

[17] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proc. NDSS*, 2016.

[18] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "DroidScribe: Classifying Android malware based on runtime behavior," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2016, pp. 252–261.

[19] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Springer, 2014, pp. 109–129.

[20] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online Malware detection with performance counters," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, 2013.

[21] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*. Springer, 2015, pp. 3–25.

[22] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. A. Ghazaleh, and D. V. Ponomarev, "EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers," *IEEE Trans. Depend. Sec. Comput.*, to be published.

[23] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digit. Invest.*, vol. 27, pp. 30–37, Dec. 2018.

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.

[25] A. Kapoor and S. Dhavale, "Control flow graph based multiclass malware detection using bi-normal separation," *Defence Sci. J.*, vol. 66, no. 2, pp. 138–145, 2016.

[26] M. H. Nguyen, D. Le Nguyen, X. M. Nguyen, and T. T. Quan, "Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning," *Comput. Secur.*, vol. 76, pp. 128–155, Jul. 2018.

[27] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Secur.*, 2011, p. 4.

[28] L. Nataraj and B. S. Manjunath, "SPAM: Signal processing to analyze malware [applications corner]," *IEEE Signal Process. Mag.*, vol. 33, no. 2, pp. 105–117, Mar. 2016.

[29] 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, p. 6, 2012.

[30] (Mar. 2018). *Sequence Intent Classification Using Hierarchical Attention Networks*. [Online]. Available: https://www.microsoft.com/developerblog/2018/03/06/sequence-intent-classification/

[31] I. Kwon and E. G. Im, "Extracting the representative API call patterns of malware families using recurrent neural network," in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, 2017, pp. 202–207.

[32] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Springer, 2006, pp. 226–248.

[33] D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious pdf files detection," in *Proc. 8th SIGSAC Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 119–130.

[34] N. Rndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 197–211.

[35] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.

[36] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 651–661.

[37] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proc. Asia Conf. Comput. Commun. Secur.*, 2018, pp. 457–468.

[38] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2017, pp. 315–327.

[39] *Cuckoo Sandbo*. Accessed: May 16, 2018. [Online]. Available:https://cuckoosandbox.org/

[40] *Openmalware*. Accessed: Apr. 5, 2018. [Online]. Available: http://malwarebenchmark.org/

[41] Virustotal. Accessed: May 18, 2018. [Online]. Available: https://www.virustotal.com/

[42] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. 26th Secur. Symp. Secur. (USENIX)*, 2017, pp. 625–642.

YUSHENG DAI received the B.S. degree in computer science and technology from Zhengzhou University, Zhengzhou, China, in 2011, and the M.S. degree in software engineering from the Beijing Institute of Technology, Beijing, China, in 2015. He is currently pursuing the Ph.D. degree with Northwestern Polytechnical University, Xi'an, China. His research interests include malware analysis and machine learning.

HUI LI received the B.S. degree in electrical engineering, the M.S. degree in circuits and systems, and the Ph.D. degree in system engineering from Northwestern Polytechnical University, Xi'an, China, in 1991, 1996, and 2006, respectively. He joined the School of Electronic Information, Northwestern Polytechnical University, in 1993, and was promoted to Associate Professor, in 2002. Since 2008, he has been a Professor with the School of Electronic Information, Northwestern Polytechnical University. His research interests include communication signal processing, massive MIMO, mmWave communications, avionics integrated system simulation, and multi-sensor information fusion.

YEKUI QIAN received the Ph.D. degree in technology of computer application from the University of Science and Technology, Nanjing, China, in 2010. He is currently an Associate Professor. His research interest includes cyberspace security.

RUIPENG YANG received the B.S. degree in computer science and technology from the Henan Institute of Finance and Economics, Zhengzhou, China, in 2005, and the M.S. degree in technology of computer application, in 2008. She is currently pursuing the Ph.D. degree with the National Digital Switching System Engineering and Technological R&D Center, Zhengzhou. Her research interests include signal processing and pattern recognition.

MIN ZHENG received the B.S. degree in cryptography engineering from the Zhengzhou Engineering and Technology Academy, Zhengzhou, China, in 1996. He is currently with the Henan Institute of Information Security Company Ltd. He is also a Senior Engineer, mainly engaged in cryptography and information security.