


Article

A Simhash-Based Integrative Features Extraction Algorithm for Malware Detection

Yihong Li ^{1,*} , Fangzheng Liu ¹, Zhenyu Du ¹ and Dubing Zhang ²

¹ Electronic Countermeasures College, National University of Defense Technology, Hefei 230031, China; yoyofangzheng@aliyun.com (F.L.); dzyu1108@163.com (Z.D.)

² 78092 troop of the PLA, Chengdu 610031, China; z_dubing@163.com

* Correspondence: 201439250428@mail.scut.edu.cn; Tel.: +86-132-95609607

Received: 9 July 2018; Accepted: 3 August 2018; Published: 14 August 2018



Abstract: In the malware detection process, obfuscated malicious codes cannot be efficiently and accurately detected solely in the dynamic or static feature space. Aiming at this problem, an integrative feature extraction algorithm based on simhash was proposed, which combines the static information e.g., API (Application Programming Interface) calls and dynamic information (such as file, registry and network behaviors) of malicious samples to form integrative features. The experiment extracts the integrative features of some static information and dynamic information, and then compares the classification, time and obfuscated-detection performance of the static, dynamic and integrated features, respectively, by using several common machine learning algorithms. The results show that the integrative features have better time performance than the static features, and better classification performance than the dynamic features, and almost the same obfuscated-detection performance as the dynamic features. This algorithm can provide some support for feature extraction of malware detection.

Keywords: malware detection; simhash; feature extraction; integrative features; static analysis; dynamic analysis

1. Introduction

In recent years, the quantity of malware has increased significantly, and new types of malware [1–3] have steadily emerged, creating severe challenges for cyberspace security. Therefore, it is critical in the field of malware detection to quickly analyze malicious samples and extract their real and effective features to form the detection model [4]. The existing sample analysis technology mainly includes static analysis and dynamic analysis. The static and dynamic features of malicious samples can be extracted separately by analysis [5,6].

The static feature is formed by analyzing the structure and format of the sample and then extracting the hash value, string information, function information, header file information, and resource description information. The technology obtains most of the malware information from the malware itself, thus the analysis results are relatively comprehensive. However, static features cannot correctly discriminate malware when the static information is packed or obfuscated or compressed [7], making it difficult for static features to express the true purpose of malware, thus affecting the accuracy of detection.

Dynamic features are the behavior of the sample execution and the features of the debug record, such as file operations, the creation and deletion of processes, and other dynamic behaviors. Since the malicious behaviors of malware at dynamic runtime can't be concealed, the extracted dynamic features provide a more realistic description than the static features. However, the extraction of dynamic features needs to be run in a virtual environment [8], which will be reset and restored to the previous

state after each malicious sample has been analyzed to ensure that the virtual environment is a real user environment. As a result, features extraction efficiency is much lower than for static features.

It can be seen from the above that it is difficult for one aspect of training and testing to make up for the other defects. Therefore, the research of integrative features aims to integrate the advantages of various types of features; this is a current trend in the development of malware detection technology [9–11]. Integrating the efficiency of static features and the authenticity of dynamic features can optimize both the performance of features and the spatio-temporal performance of features for classification training. To this end, this paper extracts integrative features of static and dynamic features based on simhash. On the basis of the simhash algorithm, the API function information of the static analysis is merged with the file, process, registry and network behavior information of the dynamic analysis to form integrative features. This algorithm can further improve the training speed of the classification model, as well as achieve better classification and obfuscated-detection results.

2. Related Work

In recent years, many researchers have proposed a variety of solutions. For instance, some detection methods are based on content signatures [12,13], which compare each sample with known malware signatures. However, the signature-based warning mechanism cannot solve the metamorphic or unidentified instances of malware [14]. Ni [15] proposed a malware classification algorithm that uses static features called MCSC (Malware Classification using Simhash and CNN), which convert the disassembled malware codes into gray images based on simhash and then identifies their families by a convolutional neural network. In addition, Idrees [16] combines permissions and intents, and extracts API calls to detect malware [17]. These methods all belong to static analysis, which means that it is the process of detecting malware without executing. Thus, it cannot reflect the behavior of malware very well, and cannot easily detect malicious code that is packed and obfuscated.

In addition to the above methods, observing dynamic behaviors and features is also usually used to detect malware [18]. For example, a sandbox is used to monitor the sample in real time and dynamically analyze the behavior by extracting the IP address, system calls, memory usage, returned values and times between consecutive system calls [19]. In addition, Shibahara [20] proposed a method for determining whether dynamic analysis should be suspended, based on network behavior, to collect malware communications efficiently and exhaustively. While dynamic analysis can offer a more comprehensive view of malware detection, the cost of building the environment and of the manual endeavors, in the process of investigation, is high. More importantly, dynamic analysis is often more complex than static analysis and requires more resources to reduce the number of false positives reported, and usually suffers from low detection rates due to lack of adequate training [21].

Therefore, we propose a method to integrate static and dynamic features to obtain new features. The remainder of this paper is organized as follows. In Section 3, we present our materials and method. Experiments and comparisons are reported in Section 4. Finally, we conclude our work in Section 5.

3. Methods

The purpose of the algorithm is to use simhash to integrate the results of static analysis and dynamic analysis of malicious samples, to form a uniquely identified hash value (i.e., the integrative feature). The process of the algorithm is shown in Figure 1.

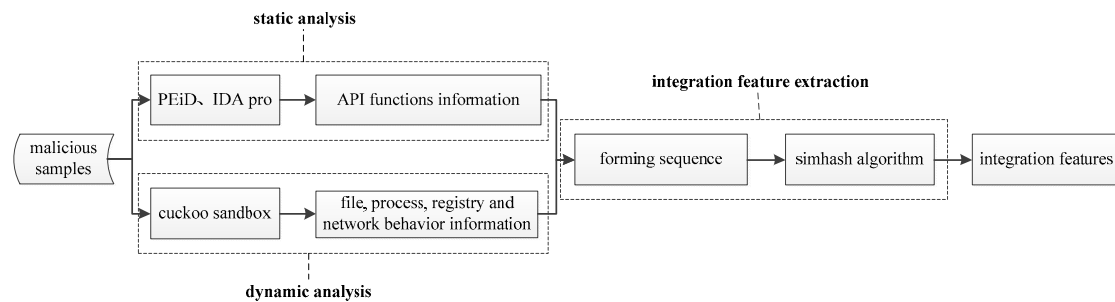


Figure 1. The process of the simhash-based integrative feature extraction algorithm. (PEiD (PE iIdentifier), IDA Pro (Interactive Disassembler Professional)).

As shown in Figure 1, the algorithm includes three sub-portions, which are a static analysis, a dynamic analysis, and a dynamic and integrative feature extraction.

In the static analysis, the first step is to judge whether the sample has a shell and, if so, use the relevant unpack tools to remove it. Next, the API call functions are obtained through IDA Pro disassembly. In the dynamic analysis, the process is to configure and debug the sample analysis environment, run the samples in the environment, and then use Process Monitor, Wireshark and other tools to capture the file, registry, process and network behaviors. In the integrative feature extraction, we use the simhash algorithm to correlate the API functions with the file, process, registry and network behaviors respectively, and finally integrate a hash binary value. The three processes are detailed below.

3.1. Static Analysis

Static analysis means obtaining the static information (e.g., structural information, format information, etc.) directly by disassembling the malicious samples. Before disassembly, it would generally detect whether the malicious samples have shells (i.e., unpacking using unpack tools) [22]. In addition, malware must interact with the system by means of the API provided by the operating system to express malicious behaviors. If the attacker does not directly use the API for system calls instead of a large amount of program code, the payload in the malicious code will be longer, which will further increase the size of the malware, in turn highlighting its malicious features and making it more easy to detect by the intrusion detection system. Although the API itself is not malicious, malware can be malicious by combining certain API functions, and these are generally uncommon in normal files such as process injection, key files alteration and deletion, etc. [23]. Therefore, this paper mainly extracts the API function information. The APIs provided by Windows systems are not only extremely large but also have different functions. From the literature, we have selected 9 API modules that may cause system security problems: advapi32.dll, kernel32.dll, ntdll.dll, Psapi.dll, rasapi32.dll, shell32.dll, user32.dll, wininet.dll and ws2_32.dll [24].

3.2. Dynamic Analysis

Dynamic analysis refers to running malicious samples in a simulated sample analysis environment (e.g., a sandbox), and then capturing the various behaviors of the samples in the run. The dynamic analysis process is more complicated than static analysis and requires longer running hours. However, the dynamic analysis process can accurately capture the behaviors [25]. Therefore, dynamic analysis is also the key to malware detection. In this paper, we used Cuckoo Sandbox [26] to monitor and capture information on four sensitive behaviors i.e. file behavior, registry behavior, process behavior and network behavior from a .json format analysis report from Cuckoo Sandbox.

3.3. Integrative Features Extraction

In dynamic analysis, the API functions are called during file read and write, memory allocation and release, and process creation and destruction. Therefore, this paper improves the simhash algorithm to mark the behavior information associated with the API with different weights, and thus extracts the integrative features. The following mainly describes the simhash algorithm and integrative process.

3.3.1. Simhash Algorithm

The simhash algorithm comes from Moses Charikar’s paper and is the core of feature extraction [27], which was originally designed to solve the deduplication tasks of hundreds of millions of web pages. The main idea is to map high-dimensional feature vectors into low-dimensional feature vectors. The implementation process is mainly divided into five steps, namely segmenting, hash, weighting, merging, and descending dimension (Figure 2).

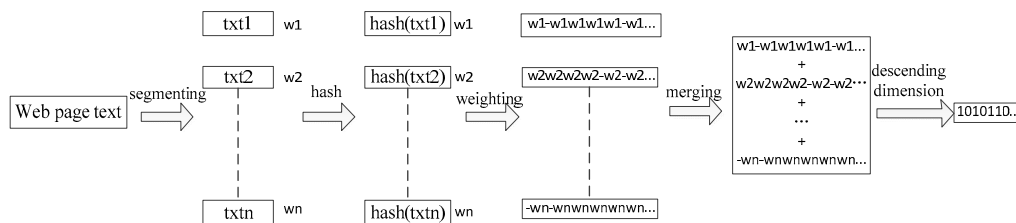


Figure 2. The process of the simhash algorithm.

3.3.2. Integrative Process

The simhash algorithm is used to integrate the dynamic and static information to form features in this paper, and some improvements are made in the process of segmenting and weighting. The process is shown in Figure 3.

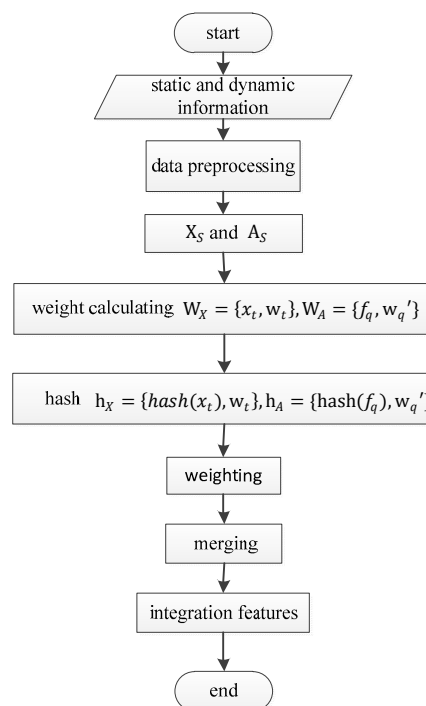


Figure 3. Integrative feature extraction.

The process is mainly divided into five steps: data preprocessing, weight calculating, hash, weighting and merging.

A. Data Preprocessing

For a certain sample S , the API function information captured by static analysis is expressed as the API call sequence, $X_s = \{x_1, x_2, \dots, x_t, \dots, x_m\}$, according to the results of monitoring by the .dll modules, where x_t represents the t -th function called, and m represents the total number of call functions. Then, dynamic analysis is used to extract the file behavior information, the process behavior information, the registry behavior information, and the network behavior information, which are respectively represented as corresponding sequences according to the execution sequence of each behavior; these are the file behavior sequence, $F_s = \{f_1, f_2, \dots, f_i, \dots, f_a\}$, the process behavior sequence, $P_s = \{p_1, p_2, \dots, p_j, \dots, p_b\}$, the registry behavior sequence, $R_s = \{r_1, r_2, \dots, r_k, \dots, r_c\}$, and the network behavior sequence, $N_s = \{n_1, n_2, \dots, n_l, \dots, n_d\}$, where: f_i is the i -th file behavior of the execution; p_j is the execution of the j -th process behavior; r_k is the execution of the k -th registry behavior; n_l is the execution of the l -th network behavior; a , b , c , and d , respectively indicate the length of each sequence of behavior; and, $A_s = \{F_s, P_s, R_s, N_s\}$.

B. Weight Calculating

Since the sequence of the API function is the same as the integrative process of each behavior sequence, the API function sequence, X_s , and the file behavior sequence, F_s , are selected as examples for explanation of weight calculating. The weights are defined as follows:

Definition 1. API function sequence weight w_t . The weight level is defined as 2. For each function, x_t , in the sequence, if x_t is related to the integrative behavior information, the weight is 2; otherwise the weight is 1.

Definition 2. File behavior sequence weight w'_i , whose weight is the number of times each behavior information, f_i , in F_s , is repeated in the sequence.

Calculating the weights of X_s and F_s as shown below:

$$W_{X_s \cup F_s} = \{(x_1, w_2), \dots, (x_t, w_t), \dots, (x_m, w_m), (f_1, w'_1), \dots, (f_i, w'_i), \dots, (f_a, w'_a)\} \quad (1)$$

C. Hash

Each of the values in X_s and F_s is hashed and mapped to a binary number of b -bits, and the calculation result is as follows:

$$H_{X_s \cup F_s} = \{(hash(x_1), w_1), \dots, (hash(x_t), w_t), \dots, (hash(x_m), w_m), (hash(f_1), w'_1), \dots, (hash(f_i), w'_i), \dots, (hash(f_a), w'_a)\} \quad (2)$$

D. Weighting

$H_{X_s \cup F_s}$ is weighted by every bit. If a bit in $hash(x_t)$ or $hash(f_i)$ is 1, then $+w_t$ or $+w'_i$; if it is 0, then $-w_t$ or $-w'_i$. For example, if the hash value is 101110, the weight is 2, and the result of the weighting calculation is $2-2222-2$. Therefore, each element in $H_{X_s \cup F_s}$ is represented as a sequence of numbers of b -bits, resulting in $H_{X_s \cup F_s, b-bits}$.

E. Merging

Each b -bits sequence in $H_{X_s \cup F_s, b-bits}$ is accumulated and merged to obtain a final b -bits sequence. Then, the b -bits sequence is normalized with a negative value taking 0 and a positive value taking 1.

Ultimately, a b-bits sequence of numbers consisting of 0 and 1 is formed, which is the integrative feature, $Z_{X_S \cup F_S}$, of X_S and F_S .

The Algorithm 1 process is as follows:

Algorithm 1 Get_Integrativefeature()

```
// Integrative feature extraction algorithm
Input: .json files of dynamic analysis, disassembled files of static analysis, samples  $n$  ( $1 \leq n \leq S$ ).
Output: four integrative features  $Z_{X_S \cup F_S}, Z_{X_S \cup R_S}, Z_{X_S \cup P_S}, Z_{X_S \cup N_S}$ .
Step 1. Let  $n = 1$ , read each line of .json file and disassembly file, capture API call sequence,  $X_S$ , file behavior sequence,  $F_S$ , registry behavior sequence,  $R_S$ , process behavior sequence,  $P_S$ , and network behavior sequence,  $N_S$ .
Step 2. Calculate the weight,  $w$ , of each of the four behavior sequences corresponding to the API call sequence, and obtain  $W_{X_S \cup F_S}, W_{X_S \cup R_S}, W_{X_S \cup P_S}, W_{X_S \cup N_S}$ .
Step 3. Calculate each hash value of  $X_S, F_S, R_S, P_S, N_S$ , and the result is represented as  $H_{X_S \cup F_S}, H_{X_S \cup R_S}, H_{X_S \cup P_S}, W_{X_S \cup N_S}$ .
Step 4. Calculate each weight value of  $H_{X_S \cup F_S}, H_{X_S \cup R_S}, H_{X_S \cup P_S}, H_{X_S \cup N_S}$  and the result is represented as  $H_{X_S \cup F_S, b\text{-bits}}, H_{X_S \cup R_S, b\text{-bits}}, H_{X_S \cup P_S, b\text{-bits}}, H_{X_S \cup N_S, b\text{-bits}}$ .
Step 5. Accumulate the sequence of each b-bit in  $H_{X_S \cup F_S, b\text{-bits}}, H_{X_S \cup R_S, b\text{-bits}}, H_{X_S \cup P_S, b\text{-bits}}, H_{X_S \cup N_S, b\text{-bits}}$ , and merge to a final b-bits sequence, and then normalize it to obtain the integrative features  $Z_{X_S \cup F_S}, Z_{X_S \cup R_S}, Z_{X_S \cup P_S}, Z_{X_S \cup N_S}$ .
```

4. Experiments and Results

4.1. Experimental Configuration

The experimental data come from VXHeavens and Malshare malware sharing websites. We collect a total of 5949 malicious samples in the form of PE (Portable Executable) files under the win32 platform in four main types: Backdoor, Trojan, Virus, and Worm (Table 1). The normal samples in the experiment are system files or various application files in .exe format, with no malicious behaviors. We use Python to implement the algorithm in this paper. The hardware and software configuration of the experiment are shown in Table 2.

Table 1. Malware Information.

Class	Amount	Average Volume (KB)	Min-Volume (Byte)	Max-Volume (KB)
Backdoor	2200	48	3500	9277
Trojan	2350	147.7	215	3800
Virus	1048	71.1	1500	1278
Worm	351	199.3	394	3087

Table 2. Configuration Information.

Property Item	Host	Virtual Host	Virtual Guest
Operating System	window7 64-bit	Ubuntu 16.04 64-bit	window7 32-bit
Running Memory	16 G	4 G	2 G
Processor	Core i5-4690	Core i5-4690	Core i5-4690
Hard Disk	1 T	120.7 G	20 G
Software Configuration	IDA pro 6.8; PEiD; VMware workstation 11; inetsim-1.2.6; cuckoo sandbox 2.0-rc2; wireshark 2.2.6; process monitor		

4.2. Experimental Design

In the experiments, correctly classified (CC), true positive (TP; the ratio of malicious samples predicted to be malicious), true negative (TN; the ratio of non-malicious samples predicted to be non-malicious), false positive (FP; the ratio of non-malicious samples predicted to be malicious),

false negative (FN; the ratio of malicious samples predicted to be non-malicious) and time (T; training time) were used as evaluation metrics. The experimental process consists of the following three parts:

4.2.1. Classification Effect Evaluation

The extracted four integrative features are represented as four-dimensional features and compared with the dynamic and static features which are not integrated. First, according to the literature [28], the N-gram algorithm is used to extract the static features from the API function information, where $N = 3$. Then, four kinds of dynamic behavior information are filtered to extract dynamic features. Finally, we use different classification algorithms to train and test the integrative features, static features and dynamic features, and evaluate the impact of different features and different classification algorithms on the classification effect. Among these, the classification algorithm is trained and tested by five common classification algorithms: NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT). The description of the selected classification algorithms is as follows.

NB is a simple technique for constructing classifiers; i.e., models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable [29]. It has the advantages of simple calculation, ability to handle multi-classification tasks, and insensitivity to missing data. However, the algorithm needs to calculate the prior probability and has a higher requirement on the form of the input data.

SGD, also known as an incremental gradient descent method, is used to optimize a differentiable objective function and a stochastic approximation of gradient descent optimization [30]. The accuracy of the algorithm is higher, but the training speed for large samples is slower.

SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. It can be used for the processing of high-dimensional features and nonlinear features, but it is less efficient for large samples and is sensitive to missing data [31].

AdaBoost is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples [32]. The algorithm can use different classification algorithms as weak classifiers, and it fully considers the weight of each classifier relative to the random forest algorithm. However, the number of weak classifiers is difficult to set, and the training takes a long time.

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set [33]. It can handle very high-dimensional features without having to make feature selections. In addition, it is fast in training and insensitive to missing features.

At last, we use the method of K-fold cross validation to train and test, where $K = 10$.

4.2.2. Obfuscated-Detection Evaluation

Obfuscated-detection refers to the ability that can obfuscate samples. First, we randomly select 500 obfuscated samples from the dataset and extract their features. Then, the 500 obfuscated samples and 500 normal samples are used as the verification set, and the three feature-trained detection models are used to detect the set.

4.2.3. Time Performance Evaluation

We compare the time it takes for different features to train the detection model by different classification algorithms.

4.3. Experimental Results

4.3.1. Classification Effect Evaluation

The training and test results of various types of features under different algorithms are shown in Tables 3–7, and the ROC curve of the algorithms is shown in Figure 4. It can be seen that the classification effect of the integrative features is almost the same as for the other two types of features, and the CC values of the three types of features are all above 80%. The experimental results are analyzed from the perspective of algorithms and features, respectively.

Table 3. The results of the naive Bayes algorithm. CC (Correctly Classified rate, CC), TP (True Positive rate, TP), TN (True Negative, TN), FP (False Positive, FP), FN (False Negative, FN), T (training Time, T).

Feature Type	Naive Bayes					
	CC	TP	TN	FP	FN	T
Static feature	0.953590	0.973	0.810	0.190	0.027	0.32s
Dynamic feature	0.897578	0.952	0.734	0.266	0.048	0.07s
Integrative feature	0.908021	0.925	0.773	0.227	0.075	0.12s

Table 4. The results of the stochastic gradient descent algorithm. CC (Correctly Classified rate, CC), TP (True Positive rate, TP), TN (True Negative, TN), FP (False Positive, FP), FN (False Negative, FN), T (training Time, T).

Feature Type	Stochastic Gradient Descent					
	CC	TP	TN	FP	FN	T
Static feature	0.983857	0.982	0.879	0.121	0.018	406.79s
Dynamic feature	0.925664	0.968	0.717	0.283	0.032	126.68s
Integrative feature	0.945034	0.965	0.807	0.193	0.035	83.08s

Table 5. The results of the support vector machine algorithm. CC (Correctly Classified rate, CC), TP (True Positive rate, TP), TN (True Negative, TN), FP (False Positive, FP), FN (False Negative, FN), T (training Time, T).

Feature Type	Support Vector Machine					
	CC	TP	TN	FP	FN	T
Static feature	0.905541	0.934	0.791	0.219	0.066	1.57s
Dynamic feature	0.848652	0.865	0.702	0.298	0.135	0.96s
Integrative feature	0.881117	0.921	0.768	0.232	0.079	1.26s

Table 6. The results of the AdaBoost algorithm. CC (Correctly Classified rate, CC), TP (True Positive rate, TP), TN (True Negative, TN), FP (False Positive, FP), FN (False Negative, FN), T (training Time, T).

Feature Type	AdaBoost					
	CC	TP	TN	FP	FN	T
Static feature	0.914763	0.936	0.782	0.218	0.064	0.27s
Dynamic feature	0.835647	0.883	0.703	0.297	0.117	0.11s
Integrative feature	0.892046	0.897	0.807	0.193	0.103	0.14s

Table 7. The results of the random trees algorithm. CC (Correctly Classified rate, CC), TP (True Positive rate, TP), TN (True Negative, TN), FP (False Positive, FP), FN (False Negative, FN), T (training Time, T).

Feature Type	Random Trees					
	CC	TP	TN	FP	FN	T
Static feature	0.954179	0.975	0.793	0.217	0.025	2.01s
Dynamic feature	0.914794	0.951	0.689	0.311	0.049	1.41s
Integrative features	0.941315	0.968	0.814	0.196	0.032	1.73s

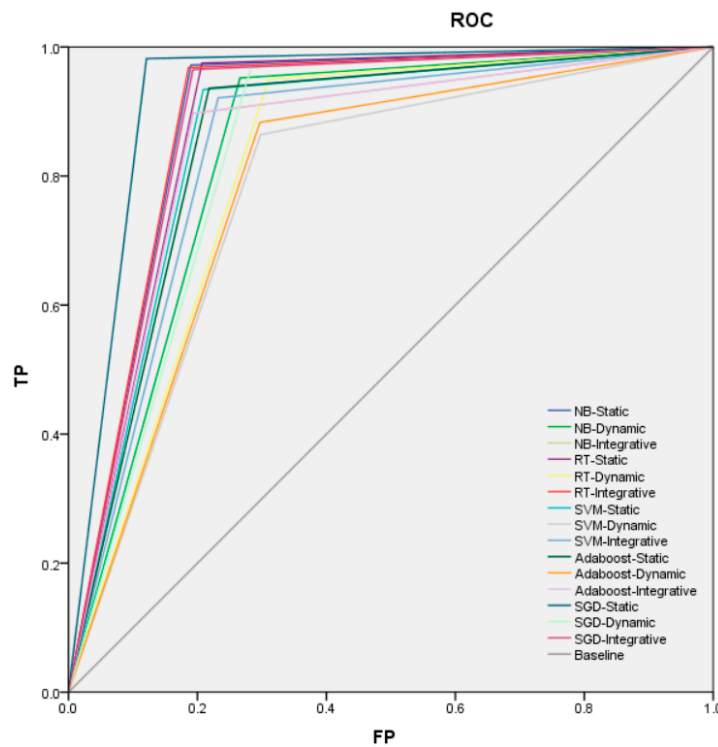


Figure 4. The ROC (Receiver Operating Characteristic curve, ROC) curve of the algorithms. TP (True Positive rate, TP), FP (False Positive, FP), NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT).

As shown in Figure 5, the CC value of the static features is higher than the other two types of features, the CC value of the dynamic features is relatively low, and the CC value of the integrative features is centered. The reason for this may be that the static information is rich and that there are more static features. Therefore, the generalization ability of the training model of the static feature is relatively high, and the CC value is increased. Some samples, e.g., samples in the .dll file format, do not have behavior information, so there are fewer dynamic features than static features and the CC values of the dynamic features is lower. The integrative features integrate the above two types of features. Although the number of features trained is large enough, the simhash algorithm is affected by the missing dynamic information during the integrative process, so that the classification effect is between the other two types of features.

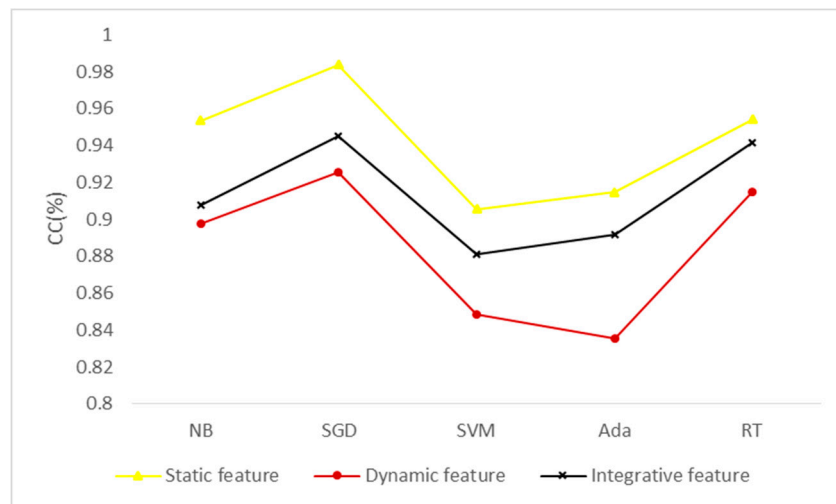


Figure 5. Classification effect of different types of features under different algorithms. CC (Correctly Classified rate, CC), NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT).

As shown in Figure 6, the model trained by the SGD algorithm has the highest CC value, followed by the RT, the SVM and Ada algorithms. The model trained by the dynamic features using the Ada algorithm has the lowest CC value. From the perspective of training time, although the SGD algorithm has the highest CC value, the training model takes a relatively long time, especially in the experiment using K-fold cross validation (i.e., approximately 40 min for 5000 samples detection), thus the algorithm is not suitable for actual detection. In summary, using the RT algorithm to train the integration integrative features is comparatively better.

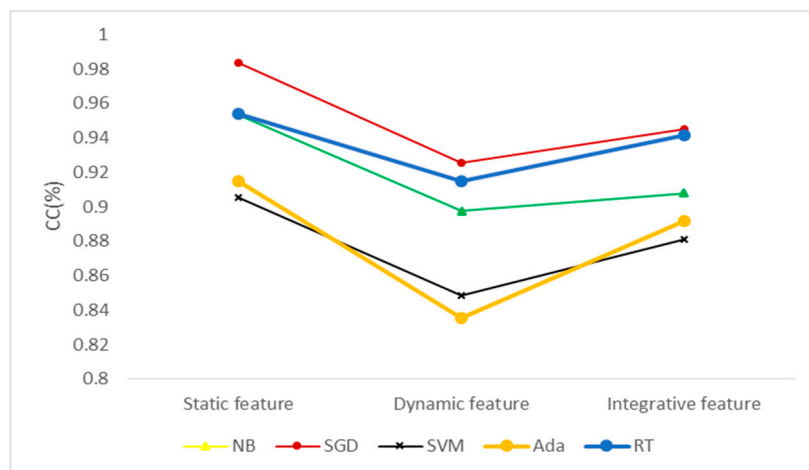


Figure 6. Classification effect of different algorithms under different types of features. CC (Correctly Classified rate, CC), NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT).

4.3.2. Obfuscated-Detection Evaluation

Table 8 and Figure 7 show the results of the obfuscated detection of the three types of features, where the DR (Detection Rate) represents the ratio of the obfuscated-detection samples to the total samples. It can be seen that the detection rate of dynamic features and integrative features is very close and better than that of static features. The reason is that it is difficult to capture the true and complete

static information of obfuscated samples in static analysis. In terms of algorithms, the detection rate using the NB, SGD or RT algorithms is higher than other algorithms.

Table 8. The obfuscated-detection results of models trained by different types of features. DR (Detection Rate, DR), NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT).

Feature Type	DR				
	NB	SGD	SVM	Ada	RT
Static feature	0.4726	0.5243	0.5137	0.4783	0.5939
Dynamic feature	0.7339	0.7595	0.5861	0.6754	0.7589
Integrative features	0.7220	0.7669	0.578	0.6812	0.7652

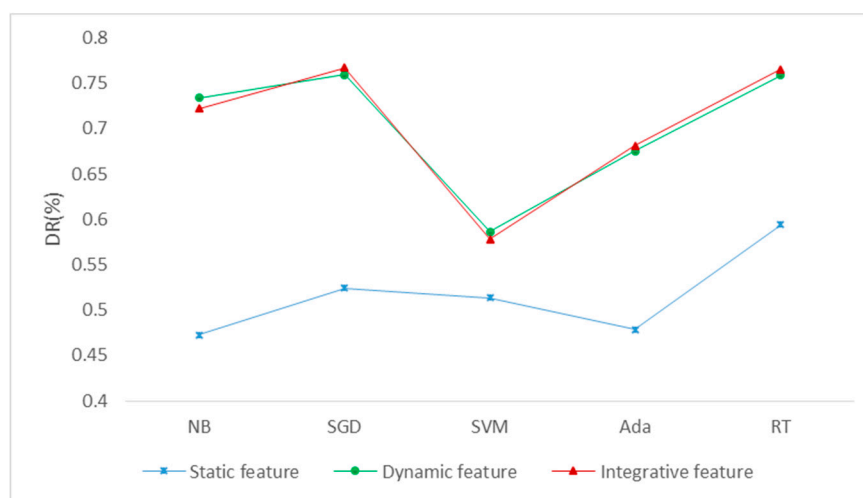


Figure 7. The obfuscated-detection results. DR (Detection Rate, DR), NB (Naive Bayes, NB), SGD (Stochastic Gradient Descent, SGD), SVM (Support Vector Machine, SVM), Ada (Adaboost, Ada) and RT (Random Trees, RT).

4.3.3. Time Performance Evaluation

As shown in Tables 3–7, the model trained with the integrative features takes the least time. The reason is that the feature uses the simhash algorithm to integrate the features into binary values, greatly reducing the storage space requirements of the training data and making training faster. However, the number of static and dynamic features is large, the storage space requirements are increased and the training speed is slow. It can be seen that the extracted integrative features in this paper have great advantages in terms of time performance.

5. Conclusions

In the field of malware detection, the features of the samples play an important role. This paper improves the simhash algorithm to integrate static and dynamic information, and proposes an extraction algorithm for integrative features. Through experimental analysis and verification, the integrative features combine the advantages of static features and dynamic features, while greatly reducing the training time of the detection model and further improving detection efficiency. However, it is also found that the amount of dynamic and static information is significant and that there is no further processing optimization, which makes it more expensive to extract integrative features using the simhash algorithm. Therefore, future research will improve the method of analyzing and

capturing filtering information to reduce the time needed for feature extraction, and integrate other static information and dynamic information.

Author Contributions: Conceptualization, Methodology, Analysis, Writing-Original Draft Preparation, Y.L.; Writing-Review and Editing, Y.L. and Z.D.; Supervision, Project Administration, Funding Acquisition, F.L.; Software, Validation, Z.D.; Investigation, Resources, D.Z.

Funding: This research was funded by the National Natural Science Foundation of China (No. U1636201).

Acknowledgments: I am very indebted to Zhang W. for his support of my work.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

PEiD v0.94 (PE iDentifier, PEiD) by snaker, Qwerton, Jibz & xineohP, U.S.

IDA Pro v6.8 by Hex-Rays Company, Liège City, Belgium.

Process Monitor v3.5 by Mark Russinovich, Sysinternals (a wholly owned subsidiary of Microsoft Corporation), Redmond, Washington, U.S.

Wireshark v2.2.1-0 by Gerald Combs, Kansas City, Missouri, U.S.

Cuckoo Sandbox v2.0.4 by Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, Mountain view city, California, U.S.

References

1. Sikorski, M.; Honig, A. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*; No Starch Press: San Francisco, CA, USA, 2012.
2. Liu, J.; Su, P.; Yang, M.; He, L.; Zhang, Y.; Zhu, X.; Lin, H. Software and Cyber Security-A Survey. *Chin. J. Softw.* **2018**, *29*, 1–20. [[CrossRef](#)]
3. Shalaginov, A.; Franke, K. Automated intelligent multinomial classification of malware species using dynamic behavioural analysis. In Proceedings of the 14th Annual Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 12–14 December 2016; pp. 70–77.
4. Lee, T.; Kim, D.; Jeong, H.; In, H.P. Risk Prediction of Malicious Code-Infected Websites by Mining Vulnerability Features. *Int. J. Secur. Appl.* **2014**, *8*, 291–294. [[CrossRef](#)]
5. Sugunan, K.; Kumar, T.G.; Dhanya, K.A. *Advances in Big Data and Cloud Computing*; Advances in Intelligent Systems and Computing In Static and Dynamic Analysis for Android Malware Detection; Rajsingh, E., Veerasamy, J., Alavi, A., Peter, J., Eds.; Springer: Singapore, 2018; pp. 147–155. [[CrossRef](#)]
6. Kang, H.; Mohaisen, A.; Kim, H. Detecting and classifying android malware using static analysis along with creator information. *Int. J. Distrib. Sens. Netw.* **2015**, *7*, 1–9. [[CrossRef](#)]
7. Zhang, M.; Raghunathan, A.; Jha, N.K. A defense framework against malware and vulnerability exploits. *Int. J. Inf. Secur.* **2014**, *13*, 439–452. [[CrossRef](#)]
8. Sujyothi, A.; Acharya, S. Dynamic Malware Analysis and Detection in Virtual Environment. *Int. J. Mod. Educ. Comput. Sci.* **2017**, *9*, 48–55. [[CrossRef](#)]
9. Cui, H.; Yu, B.; Fang, Y. Analytical method of high dimensional feature fusion for malicious classification. *Appl. Res. Comput.* **2017**, *34*, 1120–1123. [[CrossRef](#)]
10. Islam, R.; Tian, R.; Batten, L.M.; Versteeg, S. Classification of malware based on integrated static and dynamic features. *J. Netw. Comput. Appl.* **2013**, *36*, 646–656. [[CrossRef](#)]
11. Yang, H.; Zhang, Y.; Hu, Y.; Liu, Q. A malware behavior detection system of android applications based on multi-class features. *Chin. J. Comput.* **2014**, *1*, 15–27. [[CrossRef](#)]
12. Zeng, N.; Wang, Z.; Zineddin, B.; Li, Y.; Du, M.; Xiao, L.; Liu, X.; Young, T. Image-based quantitative analysis of gold immunochromatographic strip via cellular neural network approach. *IEEE Trans. Med. Imaging* **2014**, *33*, 1129–1136. [[CrossRef](#)] [[PubMed](#)]
13. Luo, X.; Zhou, M.; Leung, H.; Xia, Y.; Zhu, Q.; You, Z.; Li, S. An Incremental-and-static-combined scheme for matrix-factorization-based collaborative filtering. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 333–343. [[CrossRef](#)]

14. More, S.; Gaikwad, P. Trust-based voting method for efficient malware detection. *Procedia Comput. Sci.* **2016**, *79*, 657–667. [CrossRef]
15. Ni, S.; Qian, Q.; Zhang, R. Malware identification using visualization images and deep learning. *Comput. Secur.* **2018**. [CrossRef]
16. Idrees, F.; Rajarajan, M. Investigating the android intents and permissions for malware detection. In Proceedings of the 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications, Larnaca, Cyprus, 8–10 October 2014; pp. 354–358.
17. Rosmansyah, Y.; Dabarsyah, B. Malware detection on android smartphones using API class and machine learning. In Proceedings of the 2015 International Conference on Electrical Engineering and Informatics, Denpasar, Indonesia, 10–11 August 2015; pp. 294–297.
18. Zhu, L.; Zhao, H. Dynamical analysis and optimal control for a malware propagation model in an information network. *Neurocomputing* **2015**, *149*, 1370–1386. [CrossRef]
19. Wei, T.; Mao, C.; Jeng, A.B.; Lee, H.; Wang, H.; Wu, D. Android Malware Detection via a Latent Network Behavior Analysis. In Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, Liverpool, UK, 25–27 June 2012; pp. 1251–1258.
20. Shibahara, T.; Yagi, T.; Akiyama, M.; Chiba, D.; Yada, T. Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning. In Proceedings of the 2016 IEEE Global Communications Conference, Washington, DC, USA, 4–8 December 2016; pp. 1–7.
21. Zhu, H.; You, Z.; Zhu, Z.; Shi, W.; Chen, X.; Cheng, L. Droiddet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* **2017**, *272*, 638–646. [CrossRef]
22. Bai, L.; Pang, J.; Zhang, Y.; Fu, W.; Zhu, J. Detecting Malicious Behavior Using Critical API-Calling Graph Matching. In Proceedings of the First International Conference on Information Science & Engineering, Nanjing, China, 26–28 December 2009; pp. 1716–1719.
23. Yang, M.; Wang, S.; Ling, Z.; Liu, Y.; Ni, Z. Detection of malicious behavior in android apps through API calls and permission uses analysis. *Concurr. Comput. Pract. Exp.* **2017**, *29*, 41–72. [CrossRef]
24. Duan, X. Research on the Malware Detection Based on Windows API Call Behavior. Ph.D. Thesis, Southwest Jiaotong University, Chengdu, China, 2016.
25. Alazab, M.; Venkatraman, S.; Watters, P.; Alazab, M. Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures. In Proceedings of the Ninth Australasian Data Mining Conference, Ballarat, Australia, 1–2 December 2011; pp. 171–182.
26. Automated Malware Analysis: Cuckoo Sandbox. Available online: <http://docs.cuc-koosandbox.org> (accessed on 13 January 2018).
27. Manku, G.S.; Jain, A.; Sarma, A.D. Detecting near-duplicates for web crawling. In Proceedings of the 16th International Conference on World Wide Web, Banff, AB, Canada, 8–12 May 2007; pp. 141–150.
28. Feng, B. Malware Detection Techniques Based on Data Mining and Machine Learning. Master Thesis, Central South University, Changsha, China, 2013.
29. Naive Bayes Classifier. Available online: https://en.wikipedia.org/wiki/Naive_Bayes_classifier (accessed on 27 July 2018).
30. Song, M.; Montanari, A.; Nguyen, P. A mean field view of the landscape of two-layers neural networks. **2018**. [CrossRef]
31. Cortes, C.; Vapnik, N. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
32. AdaBoost. Available online: <https://en.wikipedia.org/wiki/AdaBoost> (accessed on 27 July 2018).
33. Ho, T. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* **1998**, *20*, 832–844. [CrossRef]

