

Classification of Malware Based on String and Function Feature Selection

Rafiqul Islam, Ronghua Tian and Lynn Batten
 School of IT
 Deakin University
 Melbourne, Australia
 {rislam, rtia, lmbatten, }@deakin.edu.au

Steve Versteeg
 CA Labs
 Melbourne, Australia
 Steve.Veersteeg@ca.com

Abstract

Anti-malware software producers are continually challenged to identify and counter new malware as it is released into the wild. A dramatic increase in malware production in recent years has rendered the conventional method of manually determining a signature for each new malware sample untenable. This paper presents a scalable, automated approach for detecting and classifying malware by using pattern recognition algorithms and statistical methods at various stages of the malware analysis life cycle. Our framework combines the static features of function length and printable string information extracted from malware samples into a single test which gives classification results better than those achieved by using either feature individually. In our testing we input feature information from close to 1400 unpacked malware samples to a number of different classification algorithms. Using k-fold cross validation on the malware, which includes Trojans and viruses, along with 151 clean files, we achieve an overall classification accuracy of over 98%.

Key words : Malware, classification, string, function length.

1. Introduction

Malicious software, malware for short, is a significant threat to the Internet as well as to the computing community. Furthermore, the anti-detection techniques employed by malware are becoming more and more sophisticated. Malware is now being driven by organized crime, with the two most common payloads being the theft of personal information and the recruitment of computers to botnets.

Until recently, developers of anti-malware software used manual means of recognizing and classifying malware. However, the enormous increase in volume now forces them to rely heavily on automatic malware analysis, detection and categorization techniques. Thus, in the last ten years or

so, researchers have searched for reliable automated malware classification methods. A substantial amount of recent research work has focused on development of tools for collecting, monitoring and run-time analysis of malware [1, 13, 2, 5, 6, 7, 8, 9, 11, 12, 16, 14]. An added complication is that current sophisticated clean software (cleanware) has many of the features of malware (using library calls, containing executables and so on); thus it is critically important to produce techniques which reliably distinguish cleanware from malware. Thus, we have included a large sample of cleanware in our experimental data.

The malware analyzed in this paper is from CA's VET Zoo (www.ca.com) and has been pre-classified using generally acceptable mechanical means. Such a classification relies on the ability to determine the significant feature or features which establish the true identification of the malware. This significant feature is referred to as 'the signature'. Many research papers have been directed at determining this signature [10, 11, 12]. The implication is that there is some unique factor which defines a piece of code. While this may be the case for a specific sample, given the many obfuscation techniques available, it is unlikely to be true for a general family; there may be several features of a piece of code which together indicate its purpose, but which separately do not definitively reveal this information.

This paper presents an automated malware classification system based on static feature selection. In previous work, we developed classification techniques based on function length [11], where feature vectors are extracted individually based on function length frequency (we refer to this technique as FLF), and then on printable string information [12] (we refer to this technique as PSI). See section 3 for details. An argument in favour of examining function features is that, while malware writers use obfuscation and polymorphic technologies, such as inserting junk bytes of instructions, instruction replacement or register replacement, these low-level assembler mnemonics would change the size and appearance of the code but not affect its overall function. Thus function features of the unpacked malware

are expected to be retained and useful in identification. In choosing strings as a feature, we noted that an obfuscated file usually does not have any strings consisting of words or sentences; packers encode almost all of the strings as non-printable or random characters. Thus those strings present should be useful in malware identification.

However, function features and string features are independent of each other and so, in classifying malware, we expect each to reinforce the other, thus giving better results than either separately. Our purpose in this paper is to test this hypothesis.

Thus, we present a combined approach in the current paper drawing on both the FLF and PSI techniques. In our experiment, we input feature information from over 1500 samples, including unpacked Trojans and viruses from the CA Zoo along with clean files from several different Windows environments. We invoked five well-known classification algorithms all from the WEKA libraries (<http://www.cs.waikato.ac.nz/ml/weka/>), including a tree-based classifier, a nearest neighbour algorithm and a statistical learning algorithm, and with each of these again tested in conjunction with the booster algorithm Adaboost. In order to perform our work more efficiently, we moved to a database environment which provided us with an integrated management interface, a standard format for binaries and better and faster interoperability with other tools and databases. We used *Ida2sql* (www.dkbza.org), a Python module which exports disassembly information from our disassembly tool IDA into the SQL schema [3].

We show in this paper that our combined classification technique achieves over 98% classification accuracy, indeed improving on both techniques separately and confirming our hypothesis. These results also strengthen the argument against the existence of a unique signature usable for malware identification.

The rest of the paper is organized as follows: background on similar classification techniques is presented in Section 2. Section 3 describes the methods used in our previous work on the function length and string methods independently. The combined classification model is described in Section 4. Section 5 describes the experimental set-up. Comparative results are presented in Section 6 and discussion and conclusions follow in Section 7.

2. Related Work

The push towards automation of malware identification and classification has led to general research into automated procedures with two approaches predominating: one based on the use of static features and the second, based on the use of dynamic features appearing when the code is executed. In this section, we first review the current static feature work, followed by the dynamic feature research.

In 2009, Leder et.al. [7] proposed an approach to classify metamorphic variants of static features. Metamorphic malware changes its code and structure from infection to infection and is missed by most virus-scanners. The authors use a static analysis technique that estimates the memory contents of the program without executing it. While their results are close to perfect, they are based on only six malware samples and cannot be applied when no estimation of the memory content is available.

Several authors have used sequences of system calls, API calls and function calls of malware in order to model its behaviour. For example, Ye et.al. [15] present a classifier using a post-processing technique based on an analysis of API execution calls. They input a large collection of executables including 35,000 malware and 15,000 cleanware samples, and using various data mining techniques they achieved close to 88% accuracy.

An intelligent instruction sequence-based malware categorization system is presented in [5], consisting of three integrated modules: feature extractor, classification and signature generator. The authors use a clustering method to classify 2029 malware samples from 407 families. Their method is similar to ours in that they use IDA Pro Disassembler to extract function calls from the unpacked malware. Their method produces close to 79% accuracy across their data set.

In [11], the authors use two aspects of functions to classify unpacked viruses by using the *k-fold* validation method: one is the length of the function as measured by the number of bytes of code in it; the other is the frequency with which function lengths occur within any particular sample of malware. The results indicate that both of these function features are significant in identifying the family to which a piece of malware belongs. The same authors also tested string information as a tool for feature extraction in [12], and claim that the string information is extremely effective for malware classification compared to previous methods. The results of both techniques along with those of several other recent papers are compared with our results in Section 5.

Finally, on the static data side, we mention the research presented in [4], which describes an architecture for automated malware classification based on massively parallel processing of common code sequences found in the static malware. Only portions of this architecture have been implemented at this time, and the cost appears to be significant.

We now turn to some recent papers which use a behavioral analysis approach. In [16], the authors trace the behavior of malware in a virtual machine environment and use these traces to extract string information on which their tests are based. From 13,223 malware samples, and using a support vector machine method, they are able to classify with

an average accuracy of 83.3%.

A behavioral model based on attribute-grammars, including semantic attributes and rules which model the behavior of malware, was developed in [6]. Their experiment tested about 200 samples of Portable Executable malware and about the same number of Visual Basic Script malware, along with 50 clean files. While their method only detected 51% of PE malware executables accurately, it performed much better on Visual Basic scripts achieving an 89% success rate.

Ahmed et.al. [1] proposed a composite scheme for malware classification that extracts statistical features from both spatial and temporal information available in run-time API calls. Using 237 API calls from six different functional categories, their tests on 416 malware and 100 cleanware files showed that their system provides an average accuracy of 98%. They also carried out a scalability analysis which shows that monitoring only memory management and file input/output API calls can provide an accuracy of 97%.

Most classification approaches assume that the malware is unpacked. Unpacking malware is a difficult and slow process which needs techniques quite different from those used for classification. Many research groups focus solely on the unpacking problem. Thus we, along with most classification researchers, assume that we are working with malware which is not packed.

In the next section we describe previous work in [11] and [12] upon which we base our current test.

3. Classification Using Static Features

Feature extraction is an important part of data classification. The general approach to the classification problem using static data is to extract a range of static features from each malware sample that can be passed to an automated classification system. In [11] and [12], the authors extracted the static features of unpacked malware based on the methods used in the FLF [11] and the PSI [12] techniques independently. This paper suggests combining the FLF and PSI features and passed it to a classification engine. The following subsections briefly describe the feature extraction process.

3.1. Function Length Frequency (FLF)

In [11], functions, as determined by IDA, form the basis of a classification system for malware. The length of the function is measured by the number of bytes of code in it. Two aspects of these functions were used: one is the frequency with which function lengths occur within any particular sample of malware; the other is function length pattern. (In the current paper, we use only the first aspect as the clas-

sification results obtained are superior to those in based on function length).

In [11], the authors extracted function length information from the Ida2DB database. For each sample a list containing the length (in bytes) for all the functions was constructed. The list was then sorted from the shortest length function to the longest, and graphed to give the function length pattern. Figure 1 displays three samples from the Robzips family and Figure 2 displays three samples from the Robknot family .

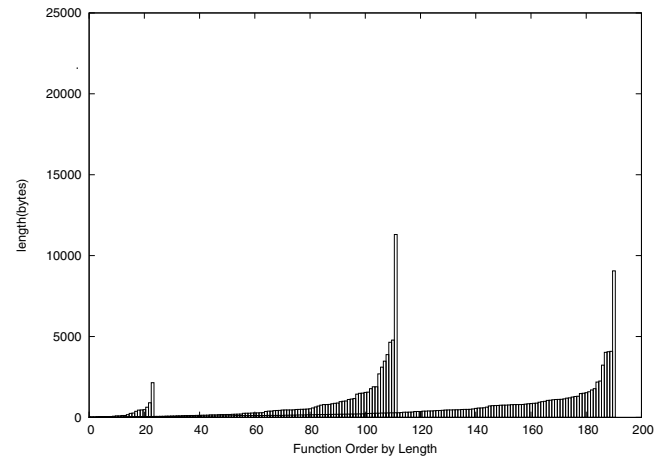


Figure 1. Samples of function length pattern for Robzips family.

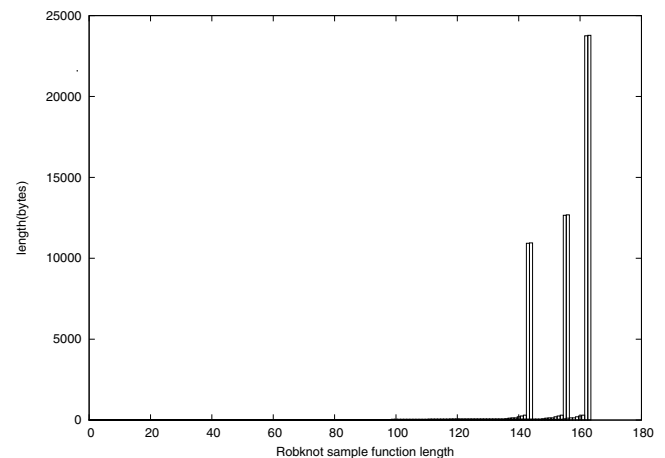


Figure 2. Samples of function length pattern for Robknot family.

The authors noticed that, within the same malware family, although the number of functions and their lengths varied, the shape of the function length pattern was similar. This motivated them to investigate whether function length

contains statistically significant information for classifying malware.

The FLF test is based on counting the number of functions in different length ranges. The function length is defined to be the number of bytes in the function as defined by IDA. An example of a function length vector, taken from the Beovens family, is (24, 38, 46, 52, 118, 122, 124, 140, 204, 650, 694, 1380). (All vectors and sets referred to in their paper, and in the current one, are ordered.) Each component in this vector represents the length of a function in the sample. We then count the number of functions in different length ranges; we say it as function length frequency (FLF).

3.2. Printable String Information (PSI)

The aim of the PSI process is to extract the printable strings from unpacked malware samples, as indicated in [12] and to analyze their occurrences in the samples. In the experiment, most the strings used for classification are not malicious in themselves but are produced by library calls. However the combination of printable strings, that is which library calls the code relies on, forms a kind of “digital fingerprint”.

Once extracted, the string information is used to create a list of all strings that occur in the database. Each string is given a specific position in the list. Each malware sample is compared with the global list and then represented by a binary vector denoting strings which the malware sample contains or not, recorded as a true/false binary value.

Figure 3 shows a sample of strings from the global list along with their frequencies in the database.

	string	frequency
1	-	1203
2	kernel32.dll\0	1145
3	LoadLibraryA\0	1074
4	GetProcAddress\0	1072
5	user32.dll\0	1036
6	ExitProcess\0	977
7	advapi32.dll\0	962
8	GetCurrentProcess\0	958
9	GetModuleHandleA\0	938
10	CloseHandle\0	933
11	GetLastError\0	873
12	GetModuleFileNameA\0	872
13	Sleep\0	863
14	WriteFile\0	860
15	GetStartupInfoA\0	838
16	CreateFileA\0	825
17	GetVersionExA\0	803
18	RegCloseKey\0	795
19	GetTickCount\0	785
20	GetStdHandle\0	732
21	TerminateProcess\0	727
22	ReadFile\0	725

Figure 3. Sample of string list

4. Proposed Classification Model

In the current paper, we merge the extracted static features using both the FLF and PSI methods from [11] and [12]. Since we use an extended data set in this paper, we repeat the disassembly process using IDA, export the disassembly analysis to a database, extract the feature vectors of both malware and clean samples, and then pass the generated vectors into the classification engine. Figure 4 shows the overview of our classification process.

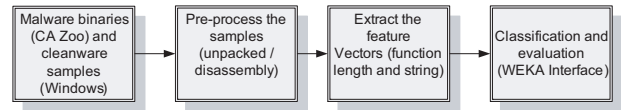


Figure 4. Static feature based classification process

4.1. Sample Preparation

The corpus of malware families used in our system was obtained from CA’s VET Zoo; the clean files were collected manually from various versions of Windows. It was necessary to unpack the malware samples before analyzing them. We used the tool VMUnpacker v1.3 to unpack most of the malware samples in this experiment.

IDA Pro V5.1 was used to disassemble the samples. Our system uses Ida2sql, a Python module [3], to export the disassembly information from IDA into an SQL database which we call Ida2DB, as shown in Figure 5. The database schema for Ida2DB is same schema used in the papers [11] and [12].

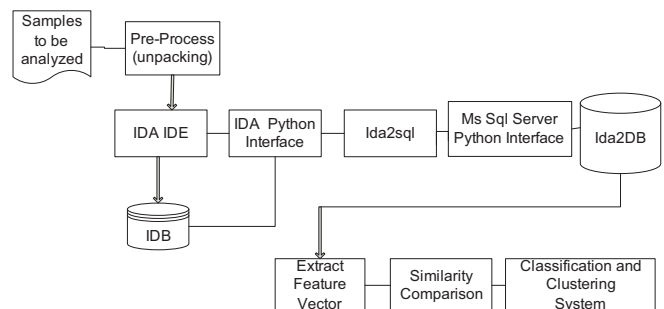


Figure 5. Block diagram of sample pre-processing

4.2 Combining the Feature Vectors

As noted earlier, the FLF and PSI features appear to contain some information which is independent of each other

and we expected each to reinforce the other, thus giving better results than either separately.

Figure 6 depicts the combined classification model. We first fetched FLF and PSI attributes from the Ida2DB database using "sql statements" which then enabled us to produce the feature vectors. Our model allows us to effectively extract large amounts of disassembly information and obtain a wide range of features of a malware sample in a swift and simple way.

In extracting the FLF features, we divide the function length scale into intervals, which we call bins, and for each sample count the frequency of functions occurring in each bin. Due to order of magnitude differences between function lengths, we used an exponential function model to represent the increase in the range covered by the bins. For example, we might count the number of functions of lengths between 1 and e bytes, the number between e and e^2 bytes, etc. In the experiment, we choose $\bar{m} = 50$ as the number of bins. This now allows us to associate a new vector of size 50 with each function length vector. We again used an exponential function over the entire spectrum of the dataset, from heights 1 to M , M being the maximum function length across the complete dataset. Assuming that this exponential function is given by $y = ae^{kx}$ where $y(0) = 1$ and $y(\bar{m}) = M$, it follows that $a = 1$, $k = \ln M/\bar{m}$, and so $y = e^{\frac{\ln M}{\bar{m}}x}$. For illustration, let $M = 500$ and $\bar{m} = 10$, which indicates that there are 10 bins and each bin ranges from $e^{(i-1)kx}$ to e^{ikx} , where $i = 1 \dots \bar{m}$. Given a sample with original vector = (2, 2, 6, 6, 8, 12, 12, 12, 12, 24, 36, 48, 56, 87, 120, 340, 469), then the standardized vector would be (0.0, 2.0, 2.0, 6.0, 0.0, 2.0, 2.0, 2.0, 0.0, 2.0).

From the extracted PSI features, we generated a string list based on the global list of strings - the ordered list of all strings occurring in our malware database. In the experiment, we chose the minimal string length to be 3, ignoring those strings of up to two bytes. Disregarding these smaller strings allowed us to significantly reduce computation.

Let $\mathbf{G} = \{s_1, s_2, s_3, \dots, s_n\}$ be the global ordered set of all strings s_i . Examples of strings were given in Figure 3. Let F represent the n families in our test, one of which contains all the cleanware samples. The notation is

$$F = \{f_1, f_2, \dots, f_n\}. \quad (1)$$

In our database, the elements of each family f_i are represented in a fashion based on the associated module_id M_j^i . For any particular family f_i having q elements and ID M_k , we can therefore write

$$f_i = \{M_1^i, M_2^i, \dots, M_q^i\} \quad (2)$$

and refer to M_q^i as the size of module_id of a particular family.

In our database M_j^i is represented as a binary vector indicating whether a particular string is present or not in the

global set of ordered strings $\mathbf{G}$, that is

$$M_j^i = (s_{j1}^i, s_{j2}^i, \dots, s_{j|G|}^i) \quad (3)$$

where

$$s_{jl}^i = \begin{cases} 0 & \text{if the } l^{th} \text{ string of } G \text{ is not in module vector } M_j^i \\ 1 & \text{otherwise} \end{cases}$$

For illustration, suppose $G = \{\text{"kernel32"}, \text{"advapi32"}, \text{"GetModuleHandleA"}, \text{"OpenProcess"}, \text{"LoadLibrary"}\}$ is the set of all strings from all samples. Let M_j^i be a member of the family f_i . Then $M_j^i = (1, 0, 1, 0, 1)$, means that the strings "advapi32" and "OpenProcess" are not present in the sample M_j^i , while the three remaining strings in G are all present.

In our experiment, we have observed that some families produce enormous sizes of features which make massive computational complexity, resulting low efficiency in the training model of classifier. We have explored those data (which are huge) and found that, a number of strings from global list occur very infrequently within the family (for example not at all or only within a handful of samples.) We assumed those strings are less significant for the family.

We have tested different reduction rates for eliminating the least occurring strings, and found 10% feature reduction gives optimum result, we say it threshold value δ_{th} . So we throw away strings whose occurrence in the family is less than δ_{th} of the total samples in the family. To select features for the family f_i based on our investigation, we then select a restricted ordered subset G_r^i , based on global set G . That is

$$G_r^i = \{g_{r1}^i, g_{r2}^i, \dots, g_{r|G_r^i|}^i\} \quad (4)$$

where g_{rj}^i are the string from the training set T_{tr} for the family f_i . The training data represents as a set of binary vector of string information, same as M_j^i as shown in equation 3, for a specific family f_i can be represents as

$$T_j^i = (t_{j1}^i, t_{j2}^i, \dots, t_{j|G_r^i|}^i) \quad (5)$$

where

$$t_{jl}^i = \begin{cases} 0 & \text{if the } l^{th} \text{ string of } G_r^i \text{ is not in module vector } M_j^i \\ 1 & \text{otherwise} \end{cases}$$

We now combine by concatenating the vectors both from the restricted PSI feature and FLF feature to produce our experimental data. Figure 7 shows the sample of combined features for sample 284 of the Robzips family.

4.3. Learning and Classification

Machine learning techniques are applied in our model for identifying the family. The basic concept of the training model of a classifier is that it employs a set of training

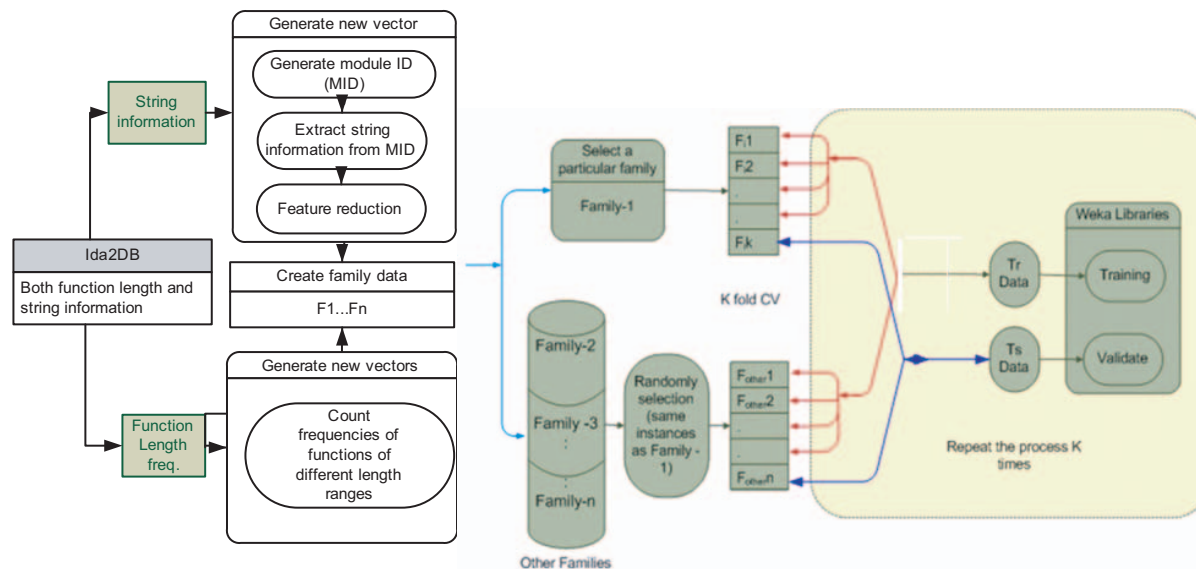


Figure 6. Block diagram of our classification model

samples of the form $\{(x_1, y_1), \dots, (x_k, y_k)\}$ for the projection of a function $f(x)$. Here the values of x are typically of the form $(x_{i,1}, x_{i,2}, \dots, x_{i,n})$, and are composed of either real or discrete values. The y values represent the expected outputs for the given x values, and are usually drawn from a discrete set of classes. Consequently, the task of a learning model involves the approximation of the function $f(x)$ to produce a classifier.

In the training model, we use a particular family and the same number of instances by randomly selected from other families. Then we split the data in K different folds and we use $(.8T_{tr} : .2T_{ts})$ technique for making k-fold cross validation. Then use these training set to train the classifier and evaluate the classification result by using the test data. The same rotating process applies for all other families.

4.4. WEKA Interface

In our system, we built an interface program for our data classification. The interface will link with Ida2DB database to collect the data for preprocessing. In our architecture we integrated WEKA into our system. WEKA is a set of data mining tools used mainly for classification and clustering. It is a collection of machine learning algorithms to perform data mining tasks, which includes decision trees and tables, rule learners, Naive Bayes, support vector machines, voted perceptrons and multi-layer perceptrons algorithms can either be applied directly to a dataset or called from your own Java code. WEKA is open source software issued under the GNU General Public License. After generating the training and test sets, we convert those data set in the WEKA format. Then use these training and test data set as input for

Robzips-284:

739,true,false,false,true,false,false,false,false,false,false,false,false,false,
lse,false,false,false,true,false,false,false,false,true,false,false,fal
se,false,false,false,false,false,true,true,true,true,true,true,true,true,
ue,true,true,true,true,true,true,true,true,true,true,true,true,false,false,
true,true,true,false,false,false,false,true,true,false,false,false,fals
e,false,false,false,false,false,false,false,false,false,false,false,fal
se,false,false,false,false,false,false,false,false,false,false,false,fal
se,false,false,false,false,true,false,false,true,false,false,true,fal
se,false,true,false,false,false,false,false,false,false,false,false,fal
se,false,false,false,false,false,false,false,false,false,false,false,0
.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
6,0,3,0,4,0,4,0,6,0,4,0,8,0,4,0,6,0,2,0,6,0,10,0,3,0,5,0,4,0,5,0,3,
0,3,0,2,0,1,0,2,0,1,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Figure 7. Combined feature vector sample for sample 284 of the Robzip family.

classifier and then call the classification library of WEKA to get our classification result. Our interface is designed in such a way that the system can select the families and the corresponding classifiers for classifying the data. In our classification model, n number of base classifiers ($n = 5$) are used and we use Meta classifier, AdaboostM1, on top of each base classifier. We build k -fold cross validation in our model to achieve better classification results.

5. Experimental Setup

This section describes the experimental setup and the methodology of our proposed classification system. We present our approach to the classification problem based on a data set of thirteen families which includes Trojans, worms, virus and clean-files. Table 1 shows the name of each family and the corresponding number of samples.

In our experiment, we tested five base classifiers: Na?ve Bayes, SVM, Random Forest, Decision Table and IB1. These were chosen to represent a spectrum of analysis techniques. In addition, the meta-classifier and booster AdaboostM1 was tested added to each of the base classifiers.

Family	# of Samples
clagger	47
robknot	101
robzipA	29
robzipB	53
alureon	56
bambo	70
beovens	144
boxed	366
emerleox	78
looked	67
agobot	340
clean	161
total	1520

Table 1. Experimental Dataset

The methodology used in our experiment is given below. We used the k -fold testing technique with $k = 5$.

1. Extract the features based on both FLF and PSI techniques.
2. Combine the features vectors and make a WEKA (arff) file.
3. Build families F_s for $s = i = 1 \dots 13$
4. Select a family F_i and split it into 5 parts.
5. Select the same instances of F_j from other families $F_j (j \neq i)$ and split into 5 parts in the same manner as for F_i

6. Build 5 sets of *Training Set Data* and *Test Set Data* in respective portions of 80% and 20%.
7. Select features from the *Training Data*.
8. Select the classifier C to be used.
9. Call WEKA libraries to train the C using the *Training Data*.
10. Evaluate the *Test Set Data*.
11. Repeat four more times.
12. Repeat for other classifiers.
13. Repeat for other classifiers.

5.1. Experimental Results

This section presents the classification outcome of the algorithms. Table 2 presents the weighted average experimental results according to base classifiers. Since not all families were of the same size, we calculated a weighted average, where each family was weighted according to the formula 6

$$\sum_{f_i}^{f_n} (WF_{si} / WF_{NS}) \quad (6)$$

Here f_i is the i th family, WF_{si} is the weight of the i th family and WF_{NS} is the total weight. Naive Bayes gives the weakest results, while the other algorithms compare very well with each other. The meta-classifier AdaBoostM1 improves on all classifiers. Based on these results, the best accuracy rate is 98.86% (AdaBoostM1 with DT). Figure 8 graphically represents the comparisons of classification accuracy with and without boosting.

As mentioned earlier that we have tested cleanware vs. malware in our experiment using our proposed combined method. It has been shown that, we have achieved over 96% accuracy (meta classifier with DT) in our testing, as indicated in table 2 and 3.

	Base Classifier				
	NB	SMO	IB1	DT	RF
W. Avg	94.31	97.77	97.63	98.15	96.69
Cleanfiles	84.51	92.4	85.64	91.04	94.4

Table 2. Classification outcome for Base classifier

Our results show that combined features, using both FLF and PSI techniques, can be used to achieve high classification accuracy for the range of methods we tested compared

Meta classifier - AdaBoost					
	NB	SMO	IB1	DT	RF
W. Avg	96.51	98.16	98.2	98.86	97.73
Cleanfiles	95.2	92.81	92.74	96.13	95.8

Table 3. Classification outcome for meta-classifier

to the previous results in [11] and [12], which were 87% and 97% respectively which tested them individually. This evidence supports our hypothesis that combining a diversity of static feature may be a powerful tool for classification of malware families as well as cleanware. Perhaps surprisingly, many of the PSI used for classifying came from library code (rather than the malicious code itself). This suggests that PSI can be used to identify which libraries the programs used. Our classification method utilises a combination of the libraries used as part of its identification of the malware family.

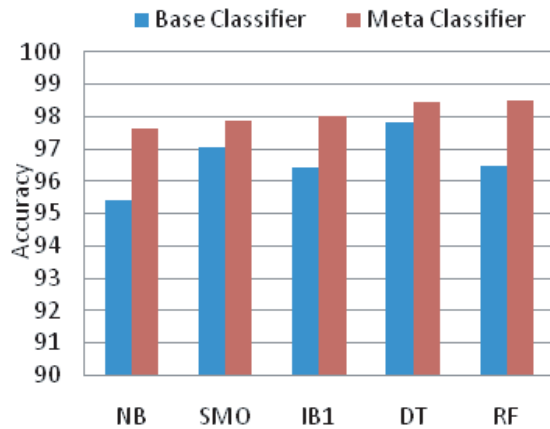


Figure 8. Comparison of classification accuracy (with and without boosting)

In Figure 9, we compare our current combined classification method with the previous individual PSI classification method [12] (which had better results than [11]). It is obvious from this figure that our present accuracy is better in all parameters compared to existing one.

In Table 4, we compare our results with some other recent work classifying large sets of malware and achieving at least 80% accuracy. Zhao, H., et al. [16] used string information to classify malware based on a total of over 13330 samples. They used training, test and validation sets and applied SVM classifiers choosing the best such

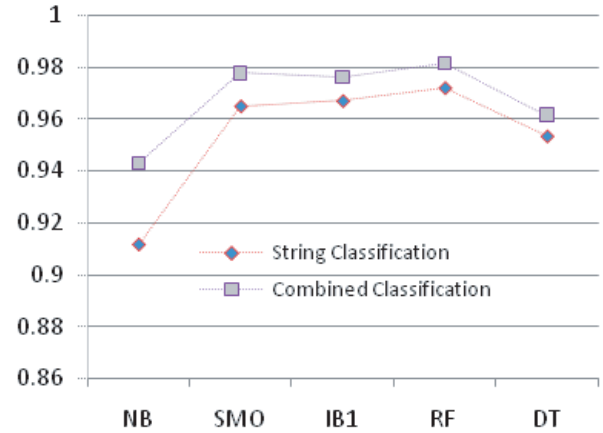


Figure 9. Comparison of current method with previous method [12].

classifier family by family. Overall performance is determined to be approximately 83.3%. Ye, Y. et.al [15], used a post-processing technique based on analysis of API execution calls. They adapt several post-processing techniques of associative classification in malware detection on approximately 35000 malicious and 15000 benign samples. Using various data mining techniques they achieved approximately 88% accuracy.

Tian et al. [11] and [12] use an analysis similar to the one described in the current paper to classify families and achieved over 87% accuracy for the FLF [11] method and 97% for the PSI method.

Finally, we include some recent work by Ahmed et al. [1] whose aim is a composite scheme for malware classification that extracts statistical features from both spatial and temporal information available in run-time API calls. Using 237 core API calls from six different functional categories, their system provides an accuracy of 98% on the average. They also carried out a scalability analysis with an accuracy of 97 %.

6. Conclusions and Future Work

This paper presents an effective malware classification technique using a diversity of static feature selection, both strings and function length, to identify and classify malware families and distinguish malware from cleanware. We chose five classifiers and an additional boosting technique (AdaboostM1) as a representative range of analytical techniques, which were then compared based on their performance. Our results demonstrated that combining string and function length features into one test provided a superior result to that of either separately. An overall classification

	Families	Samples	Accuracy	Characteristics
Ahmed F, Et.al (2009) [1]	unknown	416	98%	Behavioral
Ye, Y. et.al., (2010)[15]	unknown	35000	88%	Behavioral
Hengli Zhao, et.al., [16]	unknown	13332	83.30%	String information
Tian, R et.al. (2008) [11]	7	721	87%	Function length frequency
Tian, R et. Al (2009) [11]	13	1367	97%	Printable string information
Our method	13	1521	98.80%	Function length frequency and printable string

Table 4. Comparison of our model with similar existing work.

accuracy of 98.86% (to two decimal places) was achieved, beating all such classification methods to date. In future work we will combine and test both static and dynamic features with the target of achieving even better performance.

References

- [1] Faraz Ahmed, Haider Hameed, M. Zubair Shafiq, and Mudassar Farooq. Using spatio-temporal information in api calls with machine learning algorithms for malware detection. In *AISeC '09: Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pages 55–62, New York, NY, USA, 2009. ACM.
- [2] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, Aug 2006.
- [3] E. Carrera. 4x5:reverse engineering automation with python, 2007. <https://www.blackhat.com/presentations/bh-usa-07/Carrera/Presentation/bh-usa-07-carrera.pdf>. slides 26–32, Accessed June 12, 2008.
- [4] Timothy Daly and Luanne Burns. Concurrent architecture for automated malware classification. *Hawaii International Conference on System Sciences*, 0:1–8, 2010.
- [5] Kai Huang, Yanfang Ye, and Qinshan Jiang. Ismcs: an intelligent instruction sequence based malware categorization system. In *ASID'09: Proceedings of the 3rd international conference on Anti-Counterfeiting, security, and identification in communication*, pages 509–512, Piscataway, NJ, USA, 2009. IEEE Press.
- [6] Grégoire Jacob, Hervé Debar, and Eric Filiol. Malware detection using attribute-automata to parse abstract behavioral descriptions. *CoRR*, abs/0902.0322, 2009.
- [7] Felix Leder, Bastian Steinbock, and Peter Martini. Classification and detection of metamorphic malware using value set analysis. In *Proceedings of the 4rd International Conference on Malicious and Unwanted Software : MALWARE 2009*, pages 39–46, 2009.
- [8] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring multiple execution paths for malware analysis. *Security and Privacy, IEEE Symposium on*, 0:231–245, 2007.
- [9] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag.
- [10] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, February 2005.
- [11] R. Tian, L.M. Batten, and S.C. Versteeg. Function length as a tool for malware classification. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software : MALWARE 2008*, pages 69–76, 2008.
- [12] Ronghua Tian, Lynn Batten, Rafiqul Islam, and Steve Versteeg. An automated classification system based on the strings of trojan and virus families. In *Proceedings of the 4rd International Conference on Malicious and Unwanted Software : MALWARE 2009*, pages 23–30, 2009.
- [13] Christopher Kruegel Ulrich Bayer, , and Engin Kirda. Ttanalyze a tool for analyzing malware. In *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 2006.
- [14] Michael Vrabie, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.
- [15] Y. Ye, T. Li, Q. Jiang, and Y. Wang. CIMDS: Adapting Post-processing Techniques of Associative Classification for Malware Detection. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(3):298–307, may 2010.
- [16] Hengli Zhao, Ming Xu, Ning Zheng, Jingjing Yao, and Qiang Ho. Malicious executables classification based on behavioral factor analysis. *e-Education, e-Business, e-Management and e-Learning, International Conference on*, 0:502–506, 2010.