



Review

Classification of malware based on integrated static and dynamic features

Rafiqul Islam^{a,*}, Ronghua Tian^b, Lynn M. Batten^b, Steve Versteeg^c^a Charles Sturt University, NSW 2640, Australia^b Deakin University, Victoria 3125, Australia^c CA Labs, Melbourne, Australia

ARTICLE INFO

Article history:

Received 5 May 2012

Received in revised form

19 September 2012

Accepted 13 October 2012

Available online 2 November 2012

Keywords:

Malware classification

Static feature

Dynamic feature

ABSTRACT

Collection of dynamic information requires that malware be executed in a controlled environment; the malware unpacks itself as a preliminary to the execution process. On the other hand, while execution of malware is not needed in order to collect static information, the file must first be unpacked manually. None-the-less, if a file has been executed, it is possible to use both static and dynamic information in designing a single classification method.

In this paper, we present the first classification method integrating static and dynamic features into a single test. Our approach improves on previous results based on individual features and reduces by half the time needed to test such features separately.

Robustness to changes in malware development is tested by comparing results on two sets of malware, the first collected between 2003 and 2007, and the second collected between 2009 and 2010. When classifying the older set as compared to the entire data set, our integrated test demonstrates significantly more robustness than previous methods by losing just 2.7% in accuracy as opposed to a drop of 7%. We conclude that to achieve acceptable accuracy in classifying the latest malware, some older malware should be included in the set of data.

© 2012 Elsevier Ltd. All rights reserved.

Contents

1. Introduction	647
2. Related work	647
2.1. Static feature extraction.	647
2.1.1. Limitations of static features	648
2.2. Dynamic (run-time) feature extraction	648
2.2.1. Limitations of dynamic features	648
2.3. Machine learning techniques.	648
2.4. Motivation for the new approach	648
3. Static and dynamic feature extraction and integration	649
3.1. The data set	650
3.2. Static feature vectors	650
3.2.1. Function length frequency vectors	650
3.2.2. Printable string information (PSI) vectors	650
3.3. Dynamic feature vectors	651
3.4. The integrated feature set	651
4. Terminology and notation	652
5. The family classification experiment.	653
5.1. Classification results using the FLF method	653
5.2. Classification results using the PSI method.	653
5.3. Classification results using the dynamic method	653
5.4. Classification results using the integrated method.	653
5.5. Comparison of the integrated method with other research	653

* Corresponding author.

E-mail addresses: mislam@csu.edu.au (R. Islam), tianronghua99@gmail.com (R. Tian), lbatten@deakin.edu.au (L.M. Batten), steve.versteeg@ca.com (S. Versteeg).

6. Testing and analysis of the integrated method on old and new malware	654
6.1. Separate classifications of the old and new family sets using the integrated method	654
6.2. Performance comparison between new and old families	654
7. Conclusions	655
Funding	655
References	655

1. Introduction

The Internet has rapidly become an integral part of everyday life and our reliance on it is expected to continue to grow. However, its rapid adoption has also left it susceptible to misuse and abuse. Along with the growth of the Internet, there has been a dramatic growth in samples of malware in recent years (Ye et al., 2008). It is of major concern that despite the increasing development of anti-malware services and technologies, the volume of new malware samples continues to grow day by day (Symantec, 2009). Therefore, effective and efficient automated malware detection and response techniques are of critical importance to the computer user community.

Over the last decade, researchers have adopted a diversity of solutions in order to control malware. Much research has been conducted on developing automatic malware classification systems using data mining and machine-learning approaches (Bailey et al., 2007; Bayer et al., 2009; Gheorghescu, 2005; Gurrutxaga et al., 2008; Jiang and Zhu, 2009; Kolter and Maloof, 2004; Lee and Mody, 2006; Schultz et al., 2001; Stolfo et al., 2007; Sung et al., 2004; Tian et al., 2008; Wang et al., 2003; Ye et al., 2007). All classification approaches taken in the literature can basically be categorized into two types: (i) based on features drawn from an unpacked static version of the executable file (Santos et al., 2011; Tian et al., 2009, 2008; Wang et al., 2009; Ye et al., 2010) and (ii) based on dynamic features of the (packed) executable file (Ahmed et al., 2009; Kolbitsch et al., 2009; Zhao et al., 2010).

One of the complaints about these methods of classification is that once they are made public, a malware writer need only obfuscate the principal feature used in the classification (perhaps information about function calls or the strings present), in order to avoid detection. We address this problem in this paper by, for the first time, combining three features in our experimentation, two of which are static and one of which is dynamic, into a single integrated test.

A second complaint about standard methods of classification is that they are based on a given set of malware and may apply well to that set, but may not fare as well on more recent or future malware. In this paper, we address this point by running our experiments on two sets of executable files, one collected between 2003 and 2007 (which we refer to as the 'old family' set), and the second collected between 2009 and 2010 (which we refer to as the 'new family' set). Our results indicate that our test is robust to ongoing changes in malware and that improved results when testing new malware can be obtained if older malware are included in the data set.

In Section 2, we discuss related work in this area. In Section 3, we explain the rationale for the construction of the integrated feature vector used in the analysis. In Section 4, we describe our data set and present our classification method applying it, in Section 5, to classify malware based on our entire data set of 2939 executable files. This is followed, in Section 6, by a splitting of the data set into 'old' and 'new' files as described above, and testing the integrated approach on each set separately. Finally, in Section 7, we draw conclusions.

2. Related work

In this section, we provide background on the use of static and of dynamic features in malware classification, and on the algorithms in general use for such classification.

2.1. Static feature extraction

Conventional malware detection and classification systems are based on static features extracted from executables by reverse-engineering. A classic paper (Schultz et al., 2001) introduced the static feature extraction process based on three different types of feature: Portable Executable (PE), byte-sequence n -grams, and string features. In the PE approach, the features are extracted from dynamic link library information inside PE executables stored in Win32 PE binaries. The byte sequences (n -grams) approach uses sequences of n bytes extracted from an executable file; this type of feature was shown to yield high accuracy in detecting new malware in Schultz et al. (2001). In Abou-Assaleh et al. (2004), Kolter and Maloof (2004), and Ye et al. (2010), the authors used the n -gram technique to deal with two classification problems: (1) classification between clean and malicious executables, and (2) classification of executables as a function of their payload.

The third feature extraction approach used in Schultz et al. (2001), extracts 'strings' from the executables, which are based on text strings that are encoded in program files (Shabtai et al., 2009). This technique is shown to outperform the other two techniques in (Schultz et al., 2001). The authors of Tian et al. (2008) used only printable string information contained within the executables because they 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.

In other studies, OpCode (operational code) has been used as static information to calculate the cosine similarity between two PE executables. For example, sequence frequencies (Stolfo et al., 2007), sequences and permutations (Karim et al., 2005) and critical instruction sequences (Siddiqui et al., 2008) are used for differentiating between malware binaries. In a similar but slightly different method, the authors of Siddiqui et al. (2008) examined the OpCode frequency distribution to differentiate between malicious and clean code. However, the OpCode approach is not always feasible because some executable files cannot be disassembled properly (Bilar, 2007).

Some researchers have also used function-based feature extraction techniques (Tian et al., 2008; Shabtai et al., 2009); using this method, they extract functions from malware binaries and use them to produce various attributes to identify the file. In Tian et al. (2008), the authors used two aspects of functions to classify malware: (i) one is the function length as measured by the number of bytes of code in it and (ii) the other is the function length frequency within any particular executable file. Their

results indicate that both of these function features are significant in identifying the family to which a piece of malware belongs.

2.1.1. Limitations of static features

While static techniques are widely used for malware detection and classification, they have some major drawbacks (Menahem et al., 2009). First, they are often used to find a malware 'signature' or identity which requires a human expert and can take more time than is available to stop an attack. Secondly, the static feature approach can be easily by-passed by obfuscation methods. In Moser et al. (2007), the authors explored the limitations of the static extraction methodology and introduced a code obfuscation schema which demonstrates that static extraction alone is not enough to either detect or classify malicious code.

Thus, the inability of the static feature approach to accurately detect new forms of malicious executables shifted the focus of virus research to the determination of features that can identify malicious behavior as a process instead of by means of a unique signature.

2.2. Dynamic (run-time) feature extraction

In Wagener et al. (2008), the authors executed malware files to generate lists of API (Application Programming Interface) calls and then calculated the similarity between two API call sequences by using a similarity matrix. The run-time execution of malware files to generate usable information has also been used in the following papers: maximal common sub-graph detection by capturing system calls during the execution (Park et al., 2010); runtime behavior in the form of sequenced events with canonical format (Lin, 2008); sample of rootkits that use inline function hooking (Lobo et al., 2010). In Ahmed et al. (2009), the authors introduce a malware detection technique combining two different dynamic features (from spatial and temporal information) available in run-time API. The authors of Tian et al. (2010), proposed an automated tool running in a virtual environment to extract API call features from executables and apply pattern recognition algorithms and statistical methods to differentiate between cleanware and malware. In all of these cases, the methods give good classification accuracy but at the cost of high computational complexity.

The use of a sandbox environment is a common experimental setup when doing research in dynamic feature extraction area. While it is not the most natural environment, and so can be accused of being artificial, it facilitates rapid analysis of the samples and permits a simple re-constitution of the clean environment at the end of examination of each sample. We note that the authors of the papers (Nataraj et al., 2011; Schütt et al., 2012) use the sandbox approach. The paper (Nataraj et al., 2011) adopts a sandbox approach for isolating and analyzing malware and the automated sandbox technique assists in dealing with large quantities of malware which attacks operating systems. Using this method, they are able to examine over 100,000 malware binaries to comparatively assess their behavior. In the paper of Schütt et al. (2012), the authors compare malware detection mechanisms. They monitor the execution of JavaScript code either by running it in a sandbox or by interacting with the JavaScript browser engine.

2.2.1. Limitations of dynamic features

Dynamic analysis is time consuming as each malware sample must be executed for a certain time period and its actions logged all within a controlled environment to ensure that it cannot infect an active platform. This controlled, virtual environment is quite different from a real runtime environment and the malware may

act in different ways in the two environments resulting in an inaccurate picture of the malware in the logs.

In addition, some malware behavior is triggered only under certain conditions (via a specific command or interaction with a human, for example) and this cannot be picked up in the virtual environment. Furthermore, it has been suggested that dynamic extraction is a necessary complement to static techniques as it is significantly less vulnerable to code obfuscating transformations (Moser et al., 2007).

2.3. Machine learning techniques

In the current literature, many publications apply a machine learning approach to malware classification. Machine learning algorithms used in this area include association classifiers, support vector machines, decision tree, random forest and Naive Bayes (Abou-Assaleh et al., 2004; Moskovitch et al., 2008; Shabtai et al., 2009; Tian et al., 2008, 2010). There have also been several initiatives in automatic malware categorization using clustering techniques (Kolter and Maloof, 2004). These classification methods require a large number of training samples to build the classification models.

Our analysis of the literature on classification algorithms and our preliminary testing led to the choice of four algorithms to use in our present classification work: support vector machine (SVM), decision tree (DT), random forest (RF) and instance-based (IB1) along with boosting techniques. These algorithms represent the spectrum of major classification techniques available, based on differing approaches to the classification. (Note that while Bayesian techniques form a fifth distinctive well-known class of machine learning algorithms, our previous work (Tian et al., 2009, 2008, 2010) indicates that they perform significantly less well on this kind of data than the four techniques we shall use; hence we do not use them.)

2.4. Motivation for the new approach

In previous research work, we used the static features function length (Tian et al., 2008), printable string information (Tian et al., 2009) and dynamic log information (Tian et al., 2010) in separate classification work. To recover static features, we used IDAPro (Eagle, 2008) to disassemble the executable file. When IDA (Interactive Disassembler) disassembles unpacked malware, it identifies functions according to its own auto analysis procedure and not necessarily those returned by the malware programmer. Although it is difficult to be precise in regard to exactly what is meant by a function in this context, it is, nevertheless, a reliable and repeatable process. For each executable file, we constructed a list containing the length (in bytes) of all the functions it contained and classified all executables based on this data. The results of our tests show the true positive rate to be much higher than the false positive rate. If function length contained no information, we would expect the true positive rate and the false positive rate to be relatively equal. We therefore concluded that function length contains statistically significant information in distinguishing between families of malware, and is therefore a useful component for consideration in the classification of malware.

However, our results also indicated that it was unrealistic to expect function length information on its own to produce perfect accuracy in distinguishing between families. We then chose to classify based on printable string information. The strings utility in IDA is designed specifically to extract string content from files making it easy to collect string information. Our classification results based on printable strings demonstrated that string information can be used to achieve high classification accuracy

for the range of methods we tested. This is evidence that strings are a powerful feature for malware classification.

We then combined both function length frequency and printable string information into one classification approach (Islam et al., 2010). The combined experiment achieved over 98% classification accuracy, thus improving on both techniques separately and confirming our hypothesis that both static features are important and support each other in the classification process. These results also strengthen the argument against the existence of a unique signature usable for malware identification.

In Tian et al. (2010), we turned to the investigation of behavioral features using logs of API (application programming interface) calls. We used an automated tool running in a virtual environment to extract API call features from 1824 executables and then applied pattern recognition algorithms and statistical methods to differentiate between files. Our experimental results provided an accuracy of over 97% both in classifying malware and in distinguishing malware from cleanware.

In all experiments, we built a program to interface with the machine learning algorithms in WEKA (www.cs.waikato.ac.nz/ml/weka) for data classification.

The results of these experiments led us to believe that a single test could be developed incorporating all three features, function length, printable strings and behavioral, but with essentially the same running time as the maximum running time for each separately. In fact, the running times for the classification component of the experiments were 110 min for function length frequency features, 368 min for printed string features, 348 min for API call based features and 440 min for the integrated test presented in this paper based on base classifiers. Comparing this last value to a combined time of 826 min for running three tests separately, strongly supports

the use of this new single test as long as accuracy is at least as good as in the best of the others (see Fig. 5.2). Such an integrated test would have the additional capacity of protecting against malware writers who might focus on obfuscating part of the file, but be generally unable to obfuscate all three features without rendering the file harmless. The next section describes how we were able to integrate the features into a single test.

3. Static and dynamic feature extraction and integration

In our experiment, each of 2939 executable files, including 541 clean files, was executed and monitored using the trace tool HookMe which is designed to hook and import API functions as explained in Tian et al. (2010). The malware families used in our system were obtained from CA's (Computer Associate) VET Zoo (www.ca.com), and the clean files were collected manually from various versions of Windows but are all Win32 based. We set up the experimental environment by configuring a virtual machine on which we installed the Windows XP operating system and disabled networking. Within this virtual machine environment, we ran HookMe to monitor the execution of each file. We terminated the execution after 30 s for each file and exported the trace report outside of the virtual machine environment for further analysis. (Experiments demonstrated that 30 s were sufficient to capture all the necessary log data.)

In this section, we explain how these features were extracted and then represented in vector notation for use as input to the classifier algorithms. We describe, using examples, how and why we then combined these features into a single feature vector for use in the current experiment.

Table 3.1
Experimental data set showing old and new families.

Type	Family	Number of experimental samples
New files	Trojan	Adclicker (2009–2001 to 2010–2008)
		Bancos (2009–2001 to 2010–2007)
		Banker (2009–2001 to 2010–2006)
		Gamepass (2009–2001 to 2010–2007)
		SillyDI (2009–2001 to 2010–2008)
		Vundo (2009–01 to 2010–08)
	Worm	Frethog (2009–01 to 2010–08)
		SillyAutorun (2009–2001 to 2010–2005)
	Subtotal of new malware files	
	1517	
Old files	Trojan	Alureon (2005–2005 to 2007–2011)
		Bamboo (2003–07 to 2006–01)
		Boxed (2004–06 to 2007–09)
		Clagger (2005–2011 to 2007–2001)
		Robknot (2005–2010 to 2007–2008)
		Robzips (2006–2003 to 2007–2008)
	Virus	Looked (2003–2007 to 2006–09)
		Emerleox (2006–2011 to 2008–2011)
		Agobot (2003–2001 to 2006–2004)
	Subtotal of old malware files	
	881	
Total	Clean	541
		2939

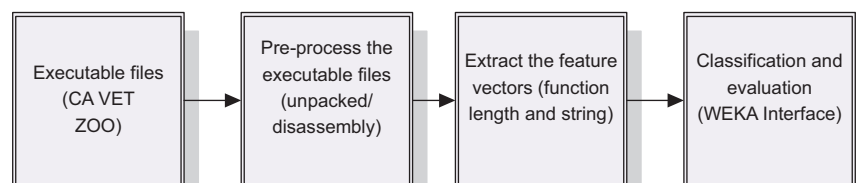


Fig. 3.1. Static feature based classification process.

3.1. The data set

Table 3.1 presents the malware used in the experiment. The associated date was found on the file label at the time the executable was captured by CA. In general, malware files are captured by anti-virus vendors within several hours of release into the wild. The malware files span a period of 8 years, but essentially fall into two time periods 2003–2007 (we refer to these files as the ‘old’ malware files) and 2009–2010 (which we refer to as the ‘new’ malware files). Only 10 malware samples were dated in the year 2008 and these all belonged to the family Emerleox; since the other samples in this family were dated from 2006 to 2007, we included all 2008 files in the ‘old’ malware set. The cleanware was collected from the Windows XP platform and was all dated between 2006 and 2010. In all, 881 of the malware samples were classified as ‘old’ using the above criterion; 1517 of the malware samples were classified as ‘new’. Note that each of the families in the data set falls entirely either into ‘old’ malware or ‘new’ malware; no family has files in each subset.

3.2. Static feature vectors

We unpacked each of the 2398 malware files by means of a command line anti-virus engine provided by CA; thus we were able to unpack the malware executables in batch mode in order to speed up the unpacking time. From the unpacked files, we then extracted the desired static features and passed the information to our automated classification system as illustrated in Fig. 3.1.

For each executable file, including malware and cleanware, we collected both function length frequency and printable string information. The following sections describe how vector representations were developed for each of these features.

3.2.1. Function length frequency vectors

The function length frequency (FLF) feature is based on counting the number of functions in different length ranges or ‘bins’. We derive a vector interpretation of an executable file based on the number of bins chosen and where the function lengths lie across the bins. For example, consider an executable file with 23 functions which have the following lengths in bytes, presented in increasing order of size: 12, 12, 12, 12, 12, 50, 50, 50, 140, 340, 420, 448, 506, 548, 828, 848, 1342, 1344, 1538, 3138, 3580, 4072, 4632. For the purposes of illustration, let us create 10 exponentially spaced bins of function length ranges. Then the distribution of lengths across the bins would be as shown in Fig. 3.2. We then capture a vector of length 10 from the last column of this figure: (0, 0, 5, 3, 1, 3, 4, 5, 2, 0). In the general case, the number of bins is fixed and so each executable is mapped to a vector of length this fixed bin size. For our general experiment we chose bin size 50 based on a global list of all function length frequencies.

3.2.2. Printable string information (PSI) vectors

In order to derive a vector representation for extracted strings for use in the classification algorithms, we again construct a global list, this time comprised of all printable strings extracted. The vector derivation method is explained by means of the following example:

Let us assume that the global string list contains 7 distinct strings: {‘GetProcAddress’, ‘RegQueryValueExW’, ‘CreateFileW’, ‘OpenFile’, ‘FindFirstFileA’, ‘FindNextFileA’, ‘CopyMemory’} for which the order is fixed. Now consider that the printable strings extracted from a particular executable file are those in the set: {‘GetProcAddress’, ‘RegQueryValueExW’, ‘CreateFileW’, ‘GetProcAddress’} listed in the order in which they were extracted. The

Length 1-2 functions	0
Length 3-7 functions	0
Length 9-21 functions	5
Length 22-59 functions	3
Length 60-166 functions	1
Length 167-464 functions	3
Length 465-1291 functions	4
Length 1292-3593 functions	5
Length 3594-9999 functions	2
Length ≥ 10000 functions	0

Fig. 3.2. Example of an FLF bins distribution.

number of strings	3
"GetProcAddress" present	true
"RegQueryValueExW" present	true
"CreateFileW" present	true
"OpenFile" present	false
"FindFirstFileA" present	false
"FindNextFileA" present	false
"CopyMemory" present	false

Fig. 3.3. Example of data used in a PSI vector.

"RegOpenKeyEx" count	0
"RegQueryValueExW" count	2
"Compositing" count	1
"RegOpenKeyExW" count	1
"0x54" count	1
"Control Panel\Desktop" count	1
"LameButtonText" count	1
"LoadLibraryW" count	1
".\UxTheme.dll" count	2
"LoadLibraryExW" count	1
"MessageBoxW" count	0

Fig. 3.4. Example of data used in a dynamic feature vector.

PSI vector for this executable file records, first of all, the total number of distinct strings in the file followed by a binary report on the presence of each string in the global list, where a ‘1’ represents the fact that the string is present and a ‘0’ that it is not. Figure 3.3 presents the corresponding data for this executable. The vector is then (3, 1, 1, 1, 0, 0, 0).

3.3. Dynamic feature vectors

After running all the executable files and logging the Windows API calls we extract API features, comprising API function names and parameters, from the log files and again construct a global list. We treat the functions and parameters as separate entities as they may separately affect the ability to detect and identify the executable.

As an example of deriving a dynamic feature vector, consider the global API feature list: {'RegOpenKeyEx', 'RegQueryValueExW', 'Compositing', 'RegOpenKeyExW', '0 × 54', 'Control Panel\Desktop', 'LameButtonText', 'LoadLibraryW', '\UxTheme.dll', 'LoadLibraryExW', 'MessageBoxW' }.

And suppose that our sample executable has the following log file:

```
2010/09/02 11:24:50.217, 180, RegQueryValueExW, Compositing
2010/09/02 11:24:50.217, 180, RegOpenKeyExW, 0 × 54, Control Panel\Desktop
2010/09/02 11:24:50.217, 180, RegQueryValueExW, LameButtonText
2010/09/02 11:24:50.217, 180, LoadLibraryW, \UxTheme.dll
2010/09/02 11:24:50.217, 180, LoadLibraryExW, \UxTheme.dll
```

The eleven items extracted are highlighted (bold italic). We then count the number of occurrences of each item in the global list and develop a vector for the sample based on the frequency of occurrence of each item as ordered by the fixed global list ordering. For this example, Fig. 3.4 gives this data. The corresponding vector is then (0, 2, 1, 1, 1, 1, 1, 2, 1, 0).

3.4. The integrated feature set

We now describe why and how we combine the feature vectors separately established in each of the above three tests into one vector.

The motivation behind combining the disparate features described in the previous sections is to prevent a malware writer from bypassing anti-virus technologies based on a single component of a malware file. Anti-virus software analyzing an executable based on function length features can be avoided by adding superfluous functions to the file in order to disguise the profile of the underlying malicious functions. Anti-virus software analyzing an executable based on features of printable strings can be by-passed by the addition of unnecessary strings to the file. API-based analysis can be avoided by the addition of useless functionalities to the executable. However, changes to all three of these components simultaneously are extremely difficult without rendering the malware useless or making it very easy to detect.

We thus propose to collect all three of the information vectors derived from FLF, PSI and API calls, and combine them into a single vector for each executable. While this appears to be a rather simplistic approach, we support it with the following points:

1. An attacker obstructing detection by altering only one of the features related to a vector representing only that feature may be viewed as introducing errors into that vector. The same errors introduced into a much longer integrated vector represents a much smaller change as a proportion of the vector length and thus is less likely to prevent correct identification of the corresponding executable.
2. Obfuscation of the entire set of features is likely to result in an outlier integrated vector based on its length or behavior and this can be detected separately.

3. Combining all feature types into a single vector allows the classifier to identify complex patterns which span multiple feature types.
4. Applying three tests in parallel requires a final decision making process in the event that there are conflicting indications about an executable. A single test gives a single output.

Comparing tables of results in Section 5, we see that on average the number of both false positives and false negatives is reduced by half in the integrated test as compared with any of the three other tests separately. This confirms the assumption underlying point 3 above.

The integrated vector is thus the concatenation of the three vectors describing separately function length frequency, printable string information and API features. Therefore for our example

Length 1-2 functions	0	FLF features
Length 3-7 functions	0	
Length 9-21 functions	5	
•		PSI features
•		
•		
number of strings	3	Dynamic features
"GetProcAddress" present	true	
"RegQueryValueExW" present	true	
•		Dynamic features
•		
•		
"RegOpenKeyEx" count	0	Dynamic features
"RegQueryValueExW" count	2	
•		
•		
•		

Fig. 3.5. Data used in generating an integrated feature vector.

Table 5.1

Weighted average classification results on the combined data set using the FLF method.

Classifier	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.297	0.121	79.057
IB1	0.172	0.137	84.506
DT	0.181	0.164	82.709
RF	0.129	0.114	87.816

Table 5.2

Weighted average classification results on the combined data set using the PSI method.

Classifiers	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.14	0.172	84.347
IB1	0.145	0.143	85.556
DT	0.119	0.149	86.59
RF	0.104	0.138	87.811

executable, the integrated feature vector would be (0, 0, 5, 3, 1, 3, 4, 5, 2, 0, 3, 1, 1, 1, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 2, 1, 0) and this data can also be seen in Fig. 3.5.

We test the effectiveness of the integrated vector approach in the classification process described in the next section. Indeed, Fig. 5.2 demonstrates the improvement of this approach over that of classification by each of the three features separately.

4. Terminology and notation

Based on the integrated vectors described in the previous section, we run a test comparing family with family, where the cleanware samples are grouped as a single family, and we compare the classification accuracy of the integrated vector method separately with those of the three feature methods discussed in Section 3.

As explained in Section 3.2.1, in order to generate vectors corresponding to function length data for each executable file, we first normalize the length of these vectors by selecting a fixed bin size. In this case, we chose the number 50. The lengths were then placed in bins by putting lengths between $e^{(j-1)\ln(M)/50}$ and $e^{j\ln(M)/50}$ in the j th bin where M is the maximum function length across the complete data set. In this case, $M=153,552$.

In determining the vectors corresponding to PSI features, we generated vectors as described in Section 3.2.2. The global string list in this case contains 1,742,490 distinct strings.

Finally, in determining the vectors corresponding to API features, we used the method described in Section 3.3. In this case the total number of API features was 72,259.

Table 5.3

Weighted average classification results on the combined data set using the dynamic method.

Classifiers	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.097	0.105	89.849
IB1	0.127	0.093	88.965
DT	0.109	0.121	88.479
RF	0.097	0.094	90.398

The integrated vectors are consequently formed as described in Section 3.4.

In Section 6, we use this same method to test the time-divided malware samples against each other. In this situation, we only use two families, the one referred to as ‘old’ and the one referred to as ‘new’. Cleanware samples are removed from the test as they span the time range in question and are unlikely to have an impact on the outcome.

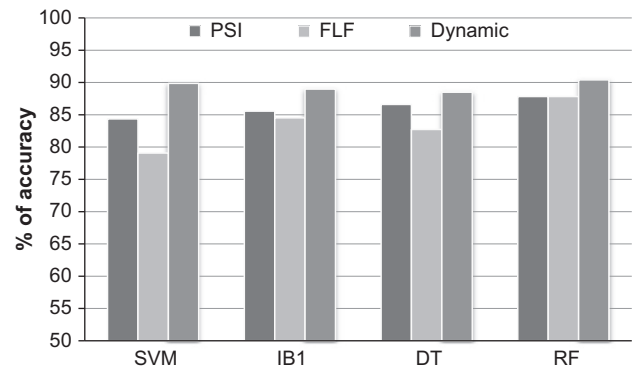


Fig. 5.1. Comparison of accuracy of the four meta-classifiers using the PSI, FLF and dynamic methods.

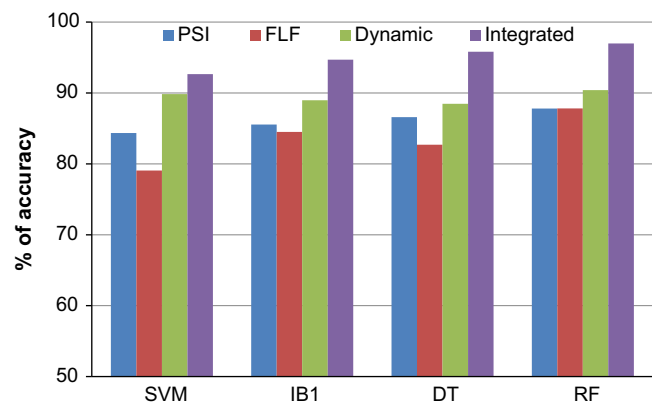


Fig. 5.2. Comparison of old and integrated methods on combined data set based on accuracy and meta-classifier.

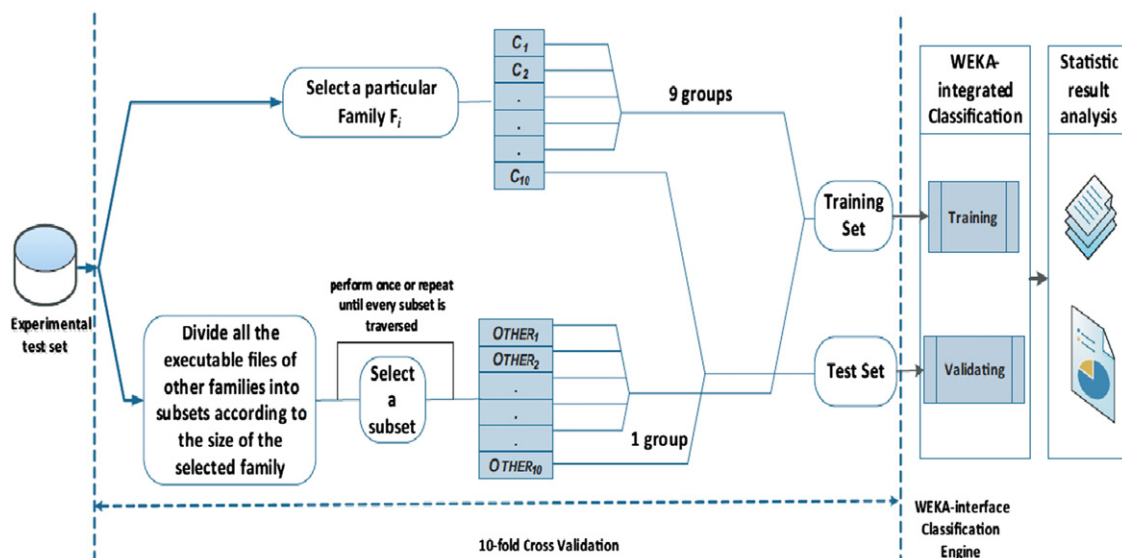


Fig. 4.1. The general classification process.

5. The family classification experiment

In the experiment we test classification accuracy based on the four base classifiers from WEKA: SVM, RF, DT and IB1 as mentioned in Section 2.3. We input the feature vectors into our classification engine for which we have written an interface. In all our experiments, we used 10-fold cross validation, as shown in Fig 4.1, as this is a standard, well understood and well accepted method of classification in a number of domains including malware analysis. Furthermore, the meta-classifier AdaboostM1 was then added to each of the base classifiers and the tests rerun. In all cases, the meta-classifiers performed better than the base classifier and so we present only the output of the meta-classifiers. The next sections present the classification results for each of the three features separately and then for the integrated test based on the combined data set of 2939 executable files.

5.1. Classification results using the FLF method

Table 5.1 shows the weighted average of the experimental results for the function length feature according to meta-classifier. We use three statistics: false positive (FP), false negative (FN) and the accuracy of detection rate (Acc). As the families are not all of the same size, we also normalized the data by calculating a weighted average, where each family was weighted according to the formula nfi/n_T , where nfi is the number of files in family f_i and n_T is the total number of files (across all families). In this paper we use the weighted average for all of our results.

The performance of our experimental study shows that meta-RF gives the best average performance compared to other classifiers.

5.2. Classification results using the PSI method

Table 5.2 shows the average classification results of the PSI method based on the combined data set. Meta-RF is, again, the best performer among the classifiers.

The classifier SVM displays the worst accuracy in both the FLF and PSI methods as indicated by Tables 5.1 and 5.2. This is perhaps

because SVM is designed to handle large data sets with a large feature space. While both feature sets for FLF and PSI are relatively small (50 and an average of 448 respectively), the feature set for the FLF test is much smaller than that for our PSI feature test which may also explain why the SVM result is better for the PSI test.

5.3. Classification results using the dynamic method

This section presents the classification performance of the dynamic method. Table 5.3 shows that the accuracy across all classifiers is better than that of the FLF and PSI tests for each meta-classifier. Once again, meta-RF achieves highest accuracy (Table 5.3).

Figure 5.1 shows the comparative results of the accuracy of the four meta-classifiers using the PSI, FLF and dynamic methods based on the 2939 files in the data set. From the figure, classifier by classifier, the dynamic method outperforms the other methods (Fig 5.1).

5.4. Classification results using the integrated method

Table 5.4 shows the family classification results, again weighted and using only the meta-classifiers. Meta-RF maintains its lead in accuracy. Note the significant improvement of the accuracy results on each classifier over those in the dynamic test despite the fact that the same dynamic data has been input. This supports our arguments in Section 3.4 for this approach. Note also the improvement in FP and FN rates, especially in the case of meta-RF.

Figure 5.2 presents a comparison of all four methods based on percentage accuracy and meta-classifier. The four methods are clearly separated in this comparison, with the FLF method showing poorest performance among the presented methods, followed by PSI, then dynamic and finally the integrated method achieving the best results. As already observed, the meta-RF classifier achieved the best overall results in family classification.

5.5. Comparison of the integrated method with other research

In this section, we compare the performance of our integrated classification approach with those of other authors. As one of our principal targets was high accuracy, we compare only with the research literature achieving 90% accuracy or better on a relatively heterogeneous (as far as we are able to tell) set of executables. Table 5.5 summarizes the comparison.

Ahmed et al. in Lobo et al. (2010) extracted statistical features from both spatial and temporal information available in run-time API calls of executables in a sandbox environment. They used a relatively small number of executable files (516) and obtained 96.3% classification accuracy. Wagener et al. (2008) tested a small set of malware, 104 files, by using similarity matrices of dynamic

Table 5.4

Weighted average of meta-classifier on the combined data set using the integrated method.

Classifiers	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.083	0.091	92.656
IB1	0.075	0.072	94.670
DT	0.060	0.100	95.677
RF	0.051	0.055	97.055

Table 5.5

Comparison of existing techniques with our integrated method.

Paper	Experimental data size	Feature extraction methods	Accuracy achieved (%)
Ahmed et al. (2009)	516	Run time API calls	96.3
Wagener et al. (2008)	104	Similarity matrix based on sequence of executed functions	~93
Bailey et al. (2007)	8000	Behavioural features based on system state changes	91.6
Ye et al. (2010)	50,000	PE instructions	93.69
Wang et al. (2009)	714	API call sequence	93.71
Moskovitch et al. (2008)	30,601	byte n -grams	95
Schultz et al. (2001)	4266	String (printable and not printable)	97.11
Menahem et al. (2009)	30,425	Integrated feature (function length, n -gram and PE)	89.35
Tian et al. (2009)	1367	Printable string information	97.5
Tian et al. (2010)	1824	API calls and API parameters	97.3
Our integrated method	2939	Integrated (FLF, PSI, API calls and API parameters) feature set	97.055

extraction technologies. They tested 104 executables and 7% of them were misclassified; their detection rate is 93%.

Bailey et al. (2007) used the technique known as system state changes, to extract behavioral features and compared over 3000 pre-classified executable files with over 3000 unclassified executable files obtaining 91.6% accuracy. In Ye et al. (2010), the authors used static API call sequences as the characterizing feature with 30,601 executable files and obtained 93.69% accuracy. Wang et al. (2009) used static API call sequences distilled from 714 Portable Executable files and pre-identified as normal (361) and virus (353) programs; they achieved 93.71% classification accuracy.

In Moskovitch et al. (2008), the authors used the text categorization process and examined the relationship between the malicious file percentage (MFP) in the test set. They found that the best mean performance of 95% is associated with a 50% MFP in the training set. Schultz et al. (2001) used n -gram and Portable Executables together with string (both printable and non-printable) features and found that string information gives the better performance of 97.11%.

Menahem et al. (2009) integrated features based on function length, n -grams and Portable Executables. They used a large sample consisting of 22,735 clean files and 7690 malware files and achieve a best average accuracy of 89.35%.

In Tian et al. (2009), printable string information is applied to a data set of 1367 executables achieving 97.5% accuracy. In Tian et al. (2010), dynamic features (API calls and their parameters) are used to 1824 executables obtaining 97.3% accuracy.

While some of the work mentioned in the above table achieves greater accuracy than does our current method, we argue that our data set is taken from a lengthy time period (8 years) and includes malware, which has had the opportunity to evolve and evade detection.

6. Testing and analysis of the integrated method on old and new malware

We now turn to the issue of robustness of the integrated test on older versus newer malware where we use the 1517 'new' malware files and 881 'old' malware files as identified in Table 3.1. Because all clean files were Windows XP based with dates spread across those of the malware, we decided not to use them in this test as they were unlikely to affect the results. The aim of this test is to determine if our classification system is robust against malware which have been given the opportunity to become adapted to old anti-virus techniques.

In order to obtain a fair comparison, we first test each family separately.

6.1. Separate classifications of the old and new family sets using the integrated method

Table 6.1 presents the 'old' family classification results by meta-classifier. In all cases, the integrated method out-performed our earlier results (Tian et al., 2009, 2008, 2010).

Table 6.2 presents the results on the new family. Here there is a significant drop in accuracy with all values being lower than respective accuracy values for the integrated method, but nevertheless being higher than in the dynamic method (Table 5.3) except for the meta-SVM classifier.

6.2. Performance comparison between new and old families

Figure 6.1 shows the comparison of classification performance between new and old malware families based on the integrated

Table 6.1

Weighted average of meta-classifier results with integrated method using 'old' family set.

Classifiers	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.003	0.017	98.924
IB1	0.005	0.015	99.168
DT	0.004	0.017	99.155
RF	0.002	0.007	99.821

Table 6.2

Weighted average of meta-classifier results with integrated method using 'new' family set.

Classifiers	Meta-classifier		
	FP	FN	Acc (%)
SVM	0.179	0.193	83.913
IB1	0.116	0.131	90.688
DT	0.098	0.132	92.393
RF	0.088	0.093	94.407

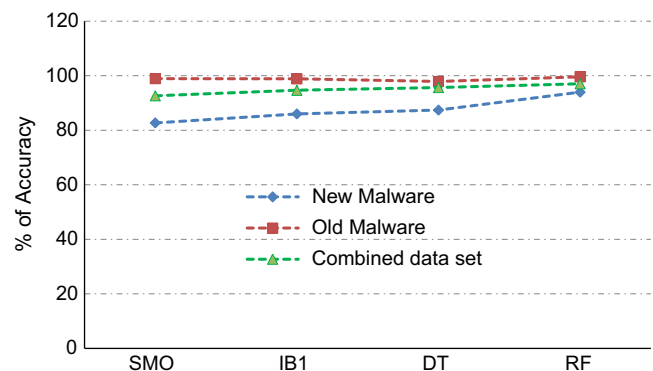


Fig. 6.1. Comparison of weighted average accuracy of old, new and combined malware using the integrated method.

test. Since the sizes of the two sets were quite different, we again applied a weighting as in Section 5.1 based on family size (no family has files across the old and new sets) to ensure a fair comparison. It is clear from the figure that the classifiers perform significantly better on the old families. However, meta-RF does remarkably well in maintaining accuracy across the board.

Although the integrated method did significantly better on the old malware set than on the new set, especially using SMO, IB1 and DT, Fig. 5.2 implies that when a mix of old and new malware is used, this discrepancy becomes much less significant. We therefore generate the comparative data in the next table based on the combined data set as opposed to the old data set (the better performer of the old and new families).

Table 6.3 summarizes the dynamic method and integrated method results. Since meta-RF was consistently the best performer, we give only the data relative to this classifier.

The integrated feature experiment had over 2% greater accuracy on the old data set than the dynamic feature experiment alone. For the combined data set, the improvement is almost 7%. These results support our claim that the integrated static and dynamic feature approach is more robust than any static or dynamic feature separately. In addition, we see that the FP values for our dynamic method on the 'old' data set and combined data set are 0.0971% and 0.0517% respectively, while the FN values

Table 6.3

Comparison of the dynamic and integrated method classification accuracy results on the combined data set.

Test	Test set	FP	FN	Acc (%)
Dynamic feature test	Old data set	0.021	0.04	97.4
	Combined data set	0.097	0.094	90.398
Integrated feature test	Old data set	0.002	0.007	99.821
	Combined data set	0.051	0.055	97.055

from the two tests are 0.0949% and 0.0559% respectively. In both cases, the latter value is close to half of the former, which means that our new method significantly improved on the false positive rate and false negative rate.

From our empirical performance presented in this Section 6.2, there is a clear trend so that we expect the results on later data to be slightly worse. While it would be optimal to be able to collect, analyze and report on the most recent malware, in reality, it takes time to work through these steps. A gap of 1–2 years between collection and results is common. To indicate that this is indeed a normal gap, we refer to the papers of Nataraj et al. (2011) and of Schütt et al. (2012).

While the paper (Nataraj et al., 2011) does not specify the dates of the more than 100,000 samples they analyze, the authors acknowledge and thank the owners of the data set Malheur consisting of over 37,000 samples, for providing it for use in their testing. This data set is at least two years old at the time the results of Nataraj et al. (2011) appear. Consequently, a large portion (at least 33%) of the data used is not ‘recent’ data in terms of the publication date of the article. This is typical of the malware classification area where it takes much time to consolidate a data set, analyze it and then examine results before publication. The authors of Schütt et al. (2012) use a data set of over 600 malicious JavaScripts, which they test on 100,000 benign web pages. All malicious data sets are taken from sources over 2 years old.

7. Conclusions

The results of Section 5 along with Fig. 5.2 indicate that the Random Forest machine learning technique is best equipped to classify our data. However, with each of RF, DT and IB1, combining the three features into a single test gives a bootstrap performance improvement of close to 9% in classification accuracy along with a corresponding improvement in false positive and false negative rates. These results support the points we make in Section 3.4 as reasons for our approach.

Considering the results of Section 6, we conclude that the age (as measured by when the executable file was first collected) of the malware used has an impact on the test results. Since our classification method is less effective on the latest malware executables as compared to the older files in our library, this demonstrates that malware continues to evolve and to deploy advanced anti-detection techniques.

Figure 6.1 and Table 6.3 indicate the importance of including both old and new malware when developing new techniques: the integrated test performed better on the combined data set than on the new malware alone, so classification of new malware should be done with old malware also present in the data set.

Funding

This work was supported by ARC Grant number LP0776260 under the auspices of the Australian Research Council and by research partner CA Technologies.

References

- Abou-Assaleh T, Cercone N, Keselj V, Sweidan R. N-gram-based detection of new malicious code. In: Proceedings of the 28th annual international computer software and applications conference—workshops and fast abstracts, vol. 02. Washington, DC, USA; IEEE Computer Society; 2004. p. 41–2.
- Ahmed F, Hameed H, Shafiq MZ, Farooq M. 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. New York, NY, USA: ACM; 2009. p. 55–62.
- Bailey M, Oberheide J, Andersen J, Mao Z, Jahanian F, Nazario J. Automated classification and analysis of internet malware. Recent Advances in Intrusion Detection 2007:178–97 [chapter].
- Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Kirda E, 2009. Scalable behavior-based malware clustering. In: Proceedings of the 16th annual network and distributed system security symposium (NDSS 2009). Available from: <http://www.isoc.org/isoc/conferences/ndss/09/pdf/11.pdf>.
- Bilar D. Opcodes as predictor for malware. Geneva, Switzerland: Inderscience Publishers; 2007 p. 156–68.
- Eagle C. The IDA pro book: the unofficial guide to the world's most popular disassembler. San Francisco, CA, USA: No Starch Press; 2008.
- Gheorghescu M. An automated classification system. In: Virus bulletin conference; October 2005. p. 294–300.
- Gurrutxaga I, Arbelaitz O, Ma Perez J, Muguerza J, Martin JI, Perona I. Evaluation of Malware clustering based on its dynamic behavior. In: John F Roddick, Jiuyong Li, Peter Christen, Paul J. Kennedy, editors. Seventh Australasian data mining conference (AusDM 2008). Glenelg, South Australia; ACS; 2008. p. 163–70.
- Islam R, Tian R, Batten L, Versteeg S. Classification of malware based on string and function feature selection. Cybercrime and Trustworthy Computing Workshop (CTC) 2010:9–17.
- Jiang X, Zhu X. Eye behavioral footprinting for self-propagating worm detection and profiling. Knowledge and Information Systems 2009;18:231–62.
- Karim M, Walenstein A, Lakhotia A, Parida L. Malware phylogeny generation using permutations of code. Journal in Computer Virology 2005;1(1):13–23.
- Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X, Wang X. Effective and efficient malware detection at the end host. In: Proceedings of the 18th conference on USENIX security symposium. Berkeley, CA, USA: USENIX Association; 2009. p. 351–66.
- Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; 2004. p. 470–8.
- Lee T, Mody JJ. Behavioral classification. In: Proceedings of the 15th European institute for computer antivirus research (EICAR 2006) annual conference; 2006. p. 1–17.
- Lin, J. On malicious software classification. In: International symposium on intelligent information technology application workshops; 2008. p. 368–71.
- Lobo D, Watters P, Wu X. RBACS rootkit behavioral analysis and classification system. In: Proceedings of the third international conference on knowledge discovery and data mining, 2010, WKDD '10; 2010. p. 75–80.
- Menahem E, Shabtai A, Rokach L, Elovici Y. Improving malware detection by applying multi-inducer ensemble. Computational Statistics and Data Analysis 2009;53(4):1483–94.
- Moser A, Kruegel C, Kirda E. Limits of static analysis for malware detection. In: Twenty-third annual computer security applications conference, 2007. ACSAC 2007; 2007. p. 421–30.
- Moskovitch R, Stopel D, Feher C, Nissim N, Elovici Y. Unknown malware detection via text categorization and the imbalance problem. ISI'2008, IEEE Press; 2008:156–61.
- Nataraj L, Yegneswaran V, Porras P, Zhang J. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the AISec '11. Chicago: ACM Press; 2011. 10 pp.
- Park Y, Reeves D, Mulukutla V, Sundaravel B. Fast malware classification by automated behavioral graph matching. In: Proceedings of the sixth annual workshop on cyber security and information intelligence research. New York, NY, USA; ACM; 2010. p. 45:1–45:4.
- Santos I, Nieves J, Bringas PG. Semi-supervised learning for unknown malware detection. In: Abraham A, Corchado JM, González S Rodríguez, de Paz Santana JF, editors. International symposium on distributed computing and artificial intelligence, advances in intelligent and soft computing, vol. 91; 2011. p. 415–22. ISBN 978-3-642-19933-2.
- Schultz MG, Eskin E, Zadok E, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE symposium on security and privacy. Washington, DC, USA; IEEE Computer Society; 2001. p. 38–49.
- Schütt K, Kloft M, Bikadorov A, Rieck K. Early detection of behavior in JavaScript code. In: Proceedings of the AISec'12. Raleigh: ACM Press; 2012. 9 pp.
- Shabtai A, Moskovitch R, Elovici Y, Glezer C. Detection of malicious code by applying machine learning classifiers on static features, a state-of-the-art survey. Oxford, UK: Elsevier Advanced Technology Publications; 2009 p. 16–29.
- Siddiqui M, Wang MC, Lee J. Data mining methods for malware detection using instruction sequences. In: Proceedings of the 26th IASTED international conference on artificial intelligence and applications. Anaheim, CA, USA: ACTA Press; 2008. p. 358–63.
- Stolfo S, Wang K, Li W-J. Towards stealthy malware detection. In: Mihai Christodorescu Somesh, Jha Douglas, Maughan Dawn, Song, Wang Cliff, editors. Malware detection. US: Springer; 2007 p. 231–49.

- Sung AH, Xu J, Chavez P, Mukkamala S. Static analyzer of vicious executables. In: ACSAC '04: proceedings of the 20th annual computer security applications conference. Washington, DC, USA: IEEE Computer Society; 2004. p. 326–34.
- Symantec. Symantec global internet security threat report trends for 2008. Technical report, Symantec; 2009.
- Tian R, Batten L, Versteeg S. Function length as a tool for malware classification. In: Proceedings of the 3rd international conference on malicious and unwanted software: MALWARE 2008; 2008. p. 69–76.
- Tian R, Batten L, Islam R, Versteeg S. An automated classification system based on the strings of Trojan and virus families. In: Proceedings of the 4th international conference on malicious and unwanted software: MALWARE 2009; 2009. p. 23–30.
- Tian R, Islam R, Batten L, Versteeg S. Differentiating malware from cleanware using behavioural analysis. In: Proceedings of the 5th international conference on malicious and unwanted software: MALWARE 2010; 2010. p. 23–30.
- Wagener G, State R, Dulaunoy A. Malware behaviour analysis. Journal in Computer Virology 2008;4(4):279–87.
- Wang C, Pang J, Zhao R, Liu, X. Using API sequence and Bayes algorithm to detect suspicious behavior. In: International conference on communication software and networks; 2009. p. 544–8.
- Wang J-H, Deng P, Fan Y-S, Jaw L-J, Liu Y-C. Virus detection using data mining techniques. Security technology 2003. In: Proceedings of the IEEE 37th annual international carnahan conference on security and technology; 2003. p. 71–6.
- Ye Y, Wang D, Li T, Ye D. IMDS intelligent malware detection system. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; 2007. p. 1043–7.
- Ye Y, Wang D, Li T, Ye D, Jiang Q. An intelligent PE-malware detection system based on association mining. Journal in Computer Virology 2008;4:323–34.
- Ye Y, Li T, Chen Y, Jiang Q. Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; 2010. p. 95–104.
- Zhao H, Xu M, Zheng N, Yao J, Ho Q. Malicious executables classification based on behavioral factor analysis. In: International conference on e-education, e-business, e-management and e-learning. IEEE Computer Society; 2010. p. 502–6.