# A Scalable Malware Classification based on Integrated Static and Dynamic Features

Tewfik Bounouh[1], Zakaria Brahimi[1], Ameer Al-Nemrat[2], and Chafika Benzaïd[3]

[1] Dept. of Computer Science, USTHB, Algérie,
[2] Architecture, Computing, and Engineering School, UEL, UK
ameer@uel.ac.uk
[3] Division Sécurité Informatique, CERIST, Algérie
cbenzaid@usthb.dz

**Abstract.** This paper presents a malware classification approach which aims to improve precision and support scalability. To this end, an hybrid approach combining both static and dynamic features is adopted. The hybrid approach has the advantage of being a complete and robust solution to evasion techniques used by malware writers.
The proposed methodology allowed achieving a very promising accuracy of 99.41% in classifying malware into families while considerably reducing the feature space compared to competing approaches in the literature.

**Keywords:** Malware classification, Static features, Dynamic features, Coarse-grained modeling

## 1 Introduction

With millions of malicious programs in the wild, and more encountered every day, malware analysis is critical for anyone who responds to computer security incidents [23].

Over the past few years, an increased interest has emerged in developing automated malware classification systems [21, 28, 14, 19, 25, 5, 6, 18, 27, 9, 20]. The existing classification systems basically rely on two analysis techniques: static and dynamic. The static analysis refers to examining the malware code without executing it, whereas the dynamic analysis consists in observing and monitoring actions performed by the malware during its execution. While static analysis provides important insights into the detection and classification of malware, its main weakness lies in coping with packing and obfuscation [17]. As a result, dynamic analysis has recently received remarkable attention as it is significantly less vulnerable to code obfuscating transformations. However, dynamic analysis has its own weaknesses. Indeed, dynamic analysis techniques monitor the malware execution in a controlled environment (e.g., a sandbox). Hence, environment-aware malware may detect the controlled environment and then prevent themselves from performing malicious behavior. Furthermore, dynamic analysis is not designed to explore all possible execution paths [17] which leads to an inaccurate picture of the malware behavior. Due to the limitations of static and dynamic analysis, relying only on one of them is insufficient to correctly classify malware. Consequently, researchers [13, 10, 20] have most recently adopted an hybrid technique which combines both static and dynamic features for better malware detection and classification. The major issues of these approaches is the size of the feature space which grows in proportion with the number of samples under examination. This might induce a scalability issue and incur high performance penalties for the classification process.

In this study, a malware classification system is developed which aims to improve precision and support scalability. With this in mind, an hybrid approach combining both static and dynamic techniques is adopted. The hybrid approach has the advantage of being a complete and robust solution to evasion techniques used by malware writers. Indeed, the static analysis overcomes the shortcomings of dynamic analysis by extracting relevant features even if the malicious software does not execute its payload during the dynamic inspection. Meanwhile, the dynamic analysis allows to collect the malicious behavior of a malware even if its code is distorted by obfuscation and protection techniques. The proposed classification framework exploits two static features; that are: printable string information due to its precision in malware detection and function length frequencies for their contribution in dissimilarity between malware families. Moreover, the dynamic features used consists in the set of actions on the system resources modeled in coarse-grained manner which allows to bound the feature space.

The rest of the paper is organized as follows. Section 2 summarizes related work in the literature. Section 3 presents the proposed malware classification framework. Section 4 discusses the experimental results. Finally, Section 5 concludes the paper.

## 2 Related Work

Over the past few years, an increased interest has emerged in developing automated malware detection and classification systems. To this end, various data mining and machine learning approaches [21, 28, 14, 19, 25, 5, 6, 18, 27, 9, 20] have been applied to categorize malware into families based on different features derived from the analysis of the malware. Indeed, malware analysis involves two fundamental techniques: static and dynamic. The static analysis refers to examining the malware code without executing it, whereas the dynamic analysis consists in observing and monitoring actions performed by the malware during its execution. In this section, we review some relevant work.

Features that are commonly gleaned from a static analysis of malware include Portable Executable (PE) header metadata such as Dynamic Link Library (DLL) [21] and API calls [28], bytes sequences (or n-grams) [21, 14, 29], Operational Codes (OpCodes) [19, 22, 24], strings [21, 25, 12], and function length and function length frequency [26]. Strings-based techniques were shown to achieve high detection and classification accuracy compared to PE and n-grams based techniques [21, 25]. Moreover, function length frequency features is significant in identifying the malware's family [13]. While static analysis provides important insights into the detection and classification of malware, its main weakness lies in coping with packing and obfuscation. In fact, Moser et al. [17] proposed an obfuscation scheme that is provably NP-hard to analyze statically.

As a result, dynamic analysis has recently received remarkable attention as it is significantly less vulnerable to code obfuscating transformations. The prominent techniques tailored to extract dynamic features are n-gram analysis [16], non-transient state changes [5], taint analysis [6, 30], system call trace analysis [18, 7, 9], and API calls [27, 8]. Dynamic approaches can further be categorized into fine-grained (e.g., [6, 30, 15]) and coarse-grained (e.g., [9]) behavior modeling approaches. The fine-grained behavior modeling approach yields precise information for executed codes, though comes with the cost of a huge feature space which might incur high performance penalties. In the other hand, coarse-grained analysis has proven its effectiveness in accurately capturing the malicious characteristics while reducing the feature space as well as the amount of noise in the extracted features [9]. In dynamic analysis techniques, malware samples typically require to be executed in a controlled environment (e.g., a sandbox). Hence, environment-aware malware may detect the controlled environment and then prevent themselves from performing malicious behavior. Furthermore, dynamic analysis is not designed to explore all possible execution paths [17] which leads to an inaccurate picture of the malware behavior.

Due to the limitations of static and dynamic analysis, relying on only one of them is insufficient to correctly classify malware. Most recently, the researchers' interest has turned to combine both static and dynamic features in the detection and classification approaches. Santos et al. [20] combine the frequency of occurrence of OpCodes sequences with the information of the execution trace of an executable. Islam et al. [13] classify binaries into malicious and benign files using function length frequency, printable string information, and API calls along with their parameters. The approach proposed in [10] distinguishes malware from cleanware based on suspicious section count, function call frequency and network and file activities along with their parameters. The major issues of these approaches is the size of the feature space which grows in proportion with the number of samples under examination. This might induce a scalability issue and incur high performance penalties for the classification process.

In this work, we adopt a hybrid approach which uses both static and dynamic characteristics. But unlike existing solutions, the proposed approach focuses on classifying malware into families rather than distinguishing between malware and cleanware. We aim to develop a classification system with a significant precision and scalability that allows the processing of large samples of malware in reduced delays. To this end, we exploit printable string information due to its precision in malware detection, function length frequencies for their contribution in dissimilarity between malware families, and the set of actions on the system resources modeled in coarse-grained manner which allows to bound the feature space.

## 3 Malware Classification Process

Our study focuses on implementing a malware classification framework. The classification system requires the succession of three distinct stages, namely: Feature extraction, feature abstraction, and then classification using a machine learning algorithm.

Firstly, static and dynamic features of each executable are collected during the extraction process. Afterward, the extracted data, which are presented in linguistic form (e.g., "Creation of file A"), will be transformed into vectors of integers during the abstraction process. These vectors represent the Cartesian coordinates of each executable in a space of $n$-dimensions. Finally, the feature vectors are passed to machine learning algorithms for malware classification.

In what follows, the different stages of the proposed system will be presented.

### 3.1 Feature Extraction

The data extraction process is the starting point of our malware classification framework. To this end, we adopt an hybrid approach which combines the extraction of static and dynamic features.

**Static Feature Extraction** From each disassembled binary code, the *printable strings* and the *function length frequency* information are collected.

**Strings**

A string is defined as a consecutive sequence of printable characters. In this work, the strings are extracted using *Hexdive* tool [3]. It is worth to mention that much of strings included in malware are irrelevant. Indeed, they are deliberately maintained in the code in order to confuse malware detection process. In order to remove noises (i.e., impertinent strings) without biasing the analysis, the *Hexdive* filter was configured to keep only strings whose the minimal length is 4 characters. Moreover, only strings related to system's actions and objects (e.g., *recv*, *ReadFile*, *GetProcAddress*) are selected. For the sake of optimizing the system's response time, the strings selected are the most dominant. That is, the strings with an apparition frequency of at least 80% in each malware family are maintained. For instance, if we have 50 malware belonging to family $A$, we will select strings whose apparition frequency is higher than 40. By applying the aforementioned selection recommendations, a global list is constructed, containing all printable strings extracted from the malware dataset.

The strings feature consists in identifying for each malware sample the total number of distinct strings found within its code and 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.

**Function Length Frequency**

The length of a function is defined as the number of bytes in the function's code. To construct the function length frequency feature, a set of length ranges are defined in order to obtain a satisfactory precision while keeping a suitable computation time. Indeed, more the number of ranges increases, more the classification precision will be increased but at the cost of an increased computation time. The function length frequency feature consists in counting the number of functions in different length ranges. The *IDA Pro* disassembler [2] was used to extract the length of functions. *IDA Pro* stands out from other disassembling tools by being programmable and by its ability to execute scripts (IDC, Python, $\cdots$ etc.) from Command Line Interface (CLI). In our case, this contributes greatly in automating the extraction process of function lengths. To this end, an IDC script was developed to retrieve lengths of all functions identified in a given malware code.

From a practical point of view, upon completion of extraction, the extracted strings and their occurrence frequencies as well as the extracted function names and their lengths are saved in a MySQL database. Once the database is filled, its content is carefully studied in order to select relevant strings and define function length ranges to be used in the abstraction phase. The selected strings and the identified function length ranges are saved in two separated files. Those files represent the constitutional dimensions of static vectors to generate during the abstraction phase.

**Dynamic Feature Extraction** Secondly, behavioral features are extracted from the reports generated after executing malicious software in a confined environment. This process is done by submitting the malware samples to Anubis [1] sandbox. Behavioral reports generated from Anubis sandbox are presented in XML format, which is by default structurally readable and platform independent suitable for a fast-paced and automated investigation.

Each report contains the execution trace of a program represented by system calls invoked to perform the program's tasks (file and registry manipulation, processes, and network communication). Recall that we adopted the approach proposed by Chandramohan *et al.* [9] to extract dynamic features from XML reports. The approach has the advantage of fixing the number of features while keeping a high accuracy rate. It is based on the assumption that the identification of malware families requires analyzing the types of security-sensitive actions performed on system resources. Thereby, the system resources and the actions performed on those resources are extracted. We limited our study to four system resources, namely file, registry, process, and network. Those resources are identified as the most security-critical system resources that are widely attacked by malware [9]. The dynamic features are based on the set of actions performed on the system resources. For sake of scalability, only non-identical set of features are kept. For instance, a set of identical actions performed on two different resource objects (e.g., files $A$ and $C$) are considered as a single malware feature. Such behavior modeling reduces the amount of noise in the malware's features space while accurately capturing the underlying malicious characteristics [9].

**File Features**

The possible actions performed by a malware on a *file* resource are: (1) *Create* action which creates a new file/directory or opens an existing file; (2) *Modify* action which modifies the file's content; (3) *Read* action which reads the file's content; (4) *Delete* action which deletes the file/directory; and (5) *Memory-mapped* action which maps the file into a virtual memory that can be shared by several processes. Thus, a malware has a maximum of $(2^5 - 1)$; that is 31 possible combinations of actions that could be performed on a file resource. Each of these combinations is modeled as a "*file*" feature. Consequently, There are 31 possible file features of a malware.

**Registry Features**

The six potential actions that a malware can perform on *registry* are as follows: (1) *Create a key* which creates a new registry key/sub-key; (2) *Delete a key* which deletes an existing registry key/sub-key; (3) *Monitor a key* which monitors the parameters of registry key/sub-key; (4) *Modify a value* which modifies the value of a registry key/sub-key; (5) *Read a value* which reads the value of a registry key/sub-key; and (6) *Delete a value* which deletes the value of a registry key/sub-key. This makes a total of $(2^6 - 1)$; that is 63 possible registry features of a malware.

**Process Features**

The potential combinations of actions on a process are formed of: (1) *Create_a_ process* which creates a process to execute a program; (2) *Delete_a_process* which terminates the process and kills all its threads; (3) *Create_a_thread* which creates a new thread to run within an executing process; (4) *Read_shared_memory* which allows the process to read the shared memory; and (5) *Write_shared_memory* which allows the process to write to the shared memory. This yields a total of $(2^5 - 1)$; that is 31 possible process features of a malware.

**Network Features**

To identify the potential actions that a malware can perform on a network, both XML reports and the packet capture (pcap) files are analyzed. For network resource, we identified 9 possible actions that are as follows: (1) *TCP_Conversation* which establishes a TCP conversation with the destination; (2) *SMTP_Conversation* which initiates a conversation with a mail server; (3) *UDP_Traffic* which allows to exchange an UDP traffic; (4) *HTTP_Traffic* which allows to exchange an HTTP traffic; (5) *FTP_Traffic* which allows to exchange an FTP traffic; (6) *Ping_Requests* which checks whether a remote target is available over the network; (7) *DNS_Query* and (8) *socket* which allow to modify the target machine's information in order to compromise its integrity; and (9) *DATA* which indicates the potential transmission and reception of data. Hence, a malware has a maximum of $(2^9 - 1)$; that is 511 possible combinations of network actions that could be performed on network resources. Consequently, There are 511 possible network features of a malware.

Unlike syntactic features, the extraction of behavioral features does not require a prior study. We just need to compare the extracted features with the set of possible combinations of each behavioral feature type. The possible combinations of actions performed on the four resources are saved in four separated files. The total size of the four files represents the constitutional dimension of dynamic vector to generate during the abstraction phase.

The result of the extraction process for each binary is a set of data including the function length distribution according to defined ranges, the strings present in code and their total number, as well as the combinations of actions performed on files, registry, processes, and network resources.

## 3.2 Data Abstraction

Data abstraction is the formalization process of the features collected during the extraction process. The choice of the formalism is closely related to the type of data supported by the classification algorithm. In our case, the classification algorithms used support Cartesian data. Hence, we chose to abstract the features as integer vectors. The vector encoding is used to represent the extracted features of a given malware as a point in an $n-$dimension space.

Firstly, dynamic features are represented with a vector of length $636 = 31 + 63 + 31 + 511$ for each malware. The first 31 bits are used for file features; the following 63 bits are used for registry features, the following 31 bits are used for process features and the last 511 bits represent the network features.

Formally, let $F$, $R$, $P$, and $N$ be the set of file features, registry features, process features and network features respectively. Let $Z^{1-31} = \{x_1 | 1 \leq x_1 \leq 31, x_1 \in \mathbb{Z}^+\}$, $Z^{32-94} = \{x_2 | 32 \leq x_2 \leq 94, x_2 \in \mathbb{Z}^+\}$, $Z^{95-125} = \{x_3 | 95 \leq x_3 \leq 125, x_3 \in \mathbb{Z}^+\}$ and $Z^{126-636} = \{x_4 | 126 \leq x_4 \leq 636, x_4 \in \mathbb{Z}^+\}$. Let $f$, $r$, $p$ and $n$ be the mapping $F \rightarrow Z^{1-31}$, $R \rightarrow Z^{32-94}$, $P \rightarrow Z^{95-125}$, $N \rightarrow Z^{126-636}$ to assign a file, registry, process and network features respectively to a positive integer in the respective range that serves as the bit position in the feature vector to represent the feature. For a malware $M$, the dynamic feature vector $V_D$ is defined as follows:

$$V_D[k] = \begin{cases} 1 \; if \; 1 \leq k \leq 31 \; and \; M \; has \; the \; feature \; x \in F, \; such \; as \; f[x] = k \\ 1 \; if \; 32 \leq k \leq 94 \; and \; M \; has \; the \; feature \; x \in R, \; such \; as \; r[x] = k \\ 1 \; if \; 94 \leq k \leq 125 \; and \; M \; has \; the \; feature \; x \in P, \; such \; as \; p[x] = k \\ 1 \; if \; 126 \leq k \leq 636 \; and \; M \; has \; the \; feature \; x \in N, \; such \; as \; n[x] = k \\ 0 \; Otherwise \end{cases} \tag{1}$$

In the same way, static features form two vectors. The first vector is for strings and the second one is for function length intervals. Let $S$ and $I$ be the set of strings and the set of function length intervals, respectively. Let $Z^{1-|S|} = \{x | 2 \leq x \leq |S|, x \in \mathbb{Z}^+\}$. Let $s$ be the mapping $S \rightarrow Z^{2-|S|}$. For a malware $M$, the string feature vector $V_{Ss}$ is defined as follows:

$$V_{Ss}[k] = \begin{cases} V_{Ss}[1] = number\ of\ strings\ found \\ 1 \qquad if\ 2 \leq k \leq |S|\ and\ M\ has\ the\ string\ x \in S,\ such\ as\ S[x] = k \\ 0 \qquad Otherwise \end{cases} \tag{2}$$

Let $Z^{1-|I|} = \{x | 1 \leq x \leq |I|, x \in \mathbb{Z}^+\}$. Let $i$ be the mapping $I \rightarrow Z^{1-|I|}$. For a malware $M$, the function length frequencies feature vector $V_{Sf}$ is defined as follows:

$$V_{Sf}[k] = \begin{cases} number\ of\ |function| \in interval\ x, \forall 1 \leq k \leq |I|; and\ x \in I,\ such\ as\ i[x] = k \\ 0\ Otherwise \end{cases} \tag{3}$$

Finally, the obtained vectors $V_D$, $V_{Ss}$, and $V_{Sf}$ are concatenated $(V_D || V_{Ss} || V_{Sf})$ to form the $V$ vector, which represents, in an $n$-dimension space, the Cartesian data of the considered malware.

## 4 Classifcation Process and Evaluation

### 4.1 Malware Dataset

The test is conducted on a set of 1515 malware, previously classified and labeled by VXheaven [4]. The malware samples span a period of 8 years (from 2007 to 2015) and belong to 26 families with more then 30 individuals per family. All malware have been designed for Windows 32 bits platform. Table 1 lists the different families and number of samples per family.

| **Malware Family** | **# of Samples** | **Malware Family** | **# of Samples** |
|---|---|---|---|
| Bakdoors | 130 | Trojan-DNSChanger-Win32 | 31 |
| DoS-Win32 | 87 | Trojan-Banker-Win32 | 43 |
| Email-Worm | 42 | Trojan-Clicker-Win32 | 90 |
| Exploit-Win32 | 39 | Trojan-DDoS-Win32 | 48 |
| Flooder-Win32 | 45 | Trojan-Downloader-Win32 | 50 |
| HackTool-Win32 | 117 | Trojan-Dropper-Win32 | 37 |
| Net-Worm-Win32 | 40 | Trojan-Proxy-Win32 | 44 |
| Packed | 55 | Trojan-Ransomware-Win32 | 42 |
| P2P-Worm-Win32 | 49 | Trojan-Spy-Win32 | 56 |
| Rootkit-Win32 | 40 | Trojan-Spy-KeyLogger | 37 |
| SpamTool-Win32 | 38 | Trojan-Spy-BotNet | 48 |
| Trojan-Win32 | 94 | Virus-Win32 | 59 |
| Trojan-Dialer-Win32 | 52 | Worm-Win32 | 102 |

Table 1: Experimental Dataset

### 4.2 Classification Process

We assessed the efficiency of the integrated approach using different classification algorithms from the Machine Learning toolbox WEKA [11]. The classification algorithms are: Support Vector Machine (SVM), Nearest Neighbor (NN), Decision Tree (DT), and Random Forest (RF). After the feature extraction and abstraction steps, the feature vectors are stored as ARFF (Attribute-Relation File Format) files and passed to WEKA for classification. It should be noted that all classifiers were evaluated by utilizing 10 fold cross-validation as it is a well accepted and the standard method of classification.

### 4.3 Evaluation Metrics

To evaluate the proposed system, True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall and Accuracy are measured. TPR is the rate of malware samples correctly classified. FPR is the rate of malware samples falsely classified. Precision is the percentage of predicted malware classes that are actually the correct malware classes. Recall is the percentage of actual malware classes predicted correctly by the model. The formulas of Precision and Recall are given by equations 4 and 5, respectively:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

Where TP is the number of malware samples correctly classified in their class, FP is the number of malware samples incorrectly classified in another class, FN is the number of malware samples incorrectly classified in their class, and TN is the number of malware samples correctly classified in other classes.

### 4.4 Experimental Results and Discussion

Table 2 shows the weighted average results when only static features are used. It is worth to note that the weighted average is used in order to deal with the imbalance in the number of samples per malware family. Meanwhile, Table 3 presents the weighted average results when both static and dynamic features are used.

The obtained results show that the hybrid method, where both static and dynamic features are used, achieves the best results with DT and SVM classifiers. Indeed, an average accuracy of above 99% is reached using DT classifier, with an improvement of 1.52% compared to using only static features. A gain of 1.52% may appear insignificant for a dataset of 1515 samples as it represents only 23 malware. However, considering a malware corpus of 1000000 samples, this gain will enhance the classification of 15200 malware samples.

| Classifier | TPR | FPR | Precision | Recall |
|---|---|---|---|---|
| NN | 0.691 | 0.014 | 0.695 | 0.691 |
| DT | 0.979 | 0.001 | 0.979 | 0.979 |
| RF | 0.964 | 0.002 | 0.964 | 0.964 |
| SVM | 0.674 | 0.015 | 0.689 | 0.674 |

Table 2: Weighted average classification results on the data set using the static method.

| Classifier | TPR | FPR | Precision | Recall |
|---|---|---|---|---|
| NN | 0.631 | 0.017 | 0.636 | 0.631 |
| DT | 0.994 | 0.001 | 0.994 | 0.994 |
| RF | 0.957 | 0.002 | 0.958 | 0.957 |
| SVM | 0.747 | 0.011 | 0.751 | 0.747 |

Table 3: Weighted average classification results on the data set using the hybrid method.

Apart from accuracy, the main advantage of the proposed hybrid approach consists in the considerable reduction of the feature space thanks to the rules used to select the most dominant static features and the coarse-grained modeling of dynamic features. To confirm this claim, we compared the number of features generated by our approach against the one proposed by islam et al. [13]. Table 4 reports the comparison results.

| | #of samples | # of features | Accuracy | Features used |
|---|---|---|---|---|
| Islam et al. [13] | 2939 | 498<br>72259<br>$Total = 72757$ | 97.05% | **Static:** Printable Strings + Function Length Frequency<br>**Dynamic:** API Calls and Parameters |
| Ours | 1515 | 289<br>636<br>$Total = 925$ | 99.41% | **Static:** Printable Strings + Function Length Frequency<br>**Dynamic:** Actions on System Resources |

Table 4: Weighted average classification results on the data set using the hybrid method.

From Table 4, we notice that the proposed approach achieves a better classification accuracy compared to [13] while considerably reducing the feature space. In fact, the feature space is reduced by 98.73% which shows the effectiveness of the coarse-grained modeling of features.

## 5 Conclusion

In this paper, we have proposed a malware classification system which aims to improve classification accuracy while supporting scalability. The accuracy goal was achieved by adopting an hybrid approach where both static and dynamic features are combined. In the other hand, the scalability objective was achieved by reducing the feature space thanks to the rules used to select the most dominant static features and the coarse-grained modeling of dynamic features..

An experimental study was conducted on a dataset of 1515 malware samples. The obtained results showed that the proposed approach reaches an average accuracy of above 99%. Moreover, the feature space was reduced by 98.73% compared to a competing approach.

## References

1. Anubis. http://anubis.seclab.tuwien.ac.at.
2. Hex-rays. ida pro. https://www.hex-rays.com/products/ida/.
3. Hexcorn ltd. hexdive. www.hexacorn.com.
4. Vxheaven. www.vxheaven.org.
5. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *In Proc. of the 10th Int. Conf. on Recent Advances in Intrusion Detection (RAID'07)*, pages 178–197. Springer-Verlag, Sep. 2007.
6. U. Bayer, P. M. Comparetti, C. Hlauscheck, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *In Proc. of the 16th Symposium on Network and Distributed System Security (NDSS'09)*, Feb. 2009.
7. R. Canzanese, S. Mancoridis, and M. Kam. Run-time classification of malicious processes using system call analysis. In *In Proc. of the 10th International Conference on Malicious and Unwanted Software (MALWARE'15)*, pages 21–28. IEEE, Oct. 2015.
8. S. Cesare, Y. Xiang, and W. Zhou. Malwise-an effective and efficient classification system for packed and polymorphic malware. *IEEE Trans. Comput.*, 62(6):1193–1206, 2013.
9. M. Chandramohan, H. B. K. Tan, and L. K. Shar. Scalable malware clustering through coarse-grained behavior modeling. In *In Proc. of the ACM SIGSOFT 20th Int. Symposium on the Foundations of Software Engineering (FSE'12)*, pages 27:1–27:4. ACM, Nov. 2012.
10. E. Gandotra, D. Bansal, and S. Sofat. Integrated framework for classification of malwares. In *In Proc. of the 7th Int. Conf. on Security of Information Networks (SIN'14)*, pages 417:417–417:422. ACM, 2014.
11. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.
12. R. Islam, R. Tian, L. Batten, and S. Versteeg. Classification of malware based on string and function feature selection. In *In Proc. of 2nd Cybercrime and Trustworthy Computing Workshop (CTC'10)*, pages 9–17. IEEE, Jul. 2010.
13. R. Islam, R. Tian, L. M. Batten, and S. Versteeg. Classification of malware based on intergrated static and dynamic features. *Journal of Network and Computer Applications*, 36:646–656, 2013.
14. J. Kolter and M. Maloof. Learning to detect malicious executables in the wild. In *In Proc. of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'04)*, pages 470–478. ACM, Aug. 2004.

15. T. Lee and J. J. Mody. Behavioral classification. In *In Proc. of the 15th Annual Conf. of the European Institute for Computer Antivirus Research (EICAR'06)*, Apr. 2006.
16. H. Mekky, A. Mohaisen, and Z.-L. Zhang. Separation of benign and malicious network events for accurate malware family classification. In *In Proc. of the IEEE Conference on Communications and Network Security (CNS'15)*, pages 125–133. IEEE, Sep. 2015.
17. A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *In Proc. of the 23rd Annual Computer Security Applications Conf. (ACSAC'07)*, pages 421–430. IEEE, Dec. 2007.
18. K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov. Learning and classification of malware behavior. In *In Proc. of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*, pages 108–125. Springer-Verlag, Jul. 2008.
19. I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas. Idea: Opcode-sequence-based malware detection. In *In Proc. of the 2nd int. Conf. on Engineering Secure Software and Systems (ESSoS'10)*, pages 35–43. Springer-Verlag, Feb. 2010.
20. I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas. Opem: A static-dynamic approach for machine learning based malware detection. In *In Proc. of the Int. Joint Conf. CISIS'12-ICEUTE'12-SOCO'12*, pages 271–280. Springer-Verlag, 2013.
21. M. Schultz, M. Eskin, E. Zadok, and F. Stolfo. Data mining methods for detection of new malicious executables. In *In Proc. of the 22nd IEEE Symposium on Security and Privacy (S&P'01)*, pages 38–49. IEEE, May 2001.
22. M. Siddiqui, M. C. Wang, and J. Lee. Data mining methods for malware detection using instruction sequences. In *In Proc. of the 26th IASTED Int. Conf. on Artificial Intelligence and Applications (AIA'08)*, pages 358–368, Feb. 2008.
23. M. Sikorski and A. Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.* No Starch Press, San Francisco, CA, USA, 1st edition, 2012.
24. S. Stolfo, K. Wang, and W.-J. Li. Towards stealthy malware detection. In *Malware Detection*, pages 231–249. Springer, 2007.
25. R. Tian, L. Batten, R. Islam, and S. Versteeg. An automated classification system based on the strings of trojan and virus families. In *In Proc. of 4th Int. Conf. on Malicious and Unwanted Software (MALWARE'09)*, pages 23–30. IEEE, Oct. 2009.
26. R. Tian, L. M. Batten, and S. C. Versteeg. Function length as a tool for malware classification. In *In Proc. of the 3rd Int. Conf. on Malicious and Unwanted Software (MALWARE'08)*, pages 69–76. IEEE, Oct. 2008.
27. R. Tian, R. Islam, L. Batten, and S. Versteeg. Differentiating malware from cleanware using behavioural analysis. In *In Proc. of the 5th Int. Conf. on Malicious and Unwanted Software (MALWARE'10)*, pages 23–30. IEEE, Oct. 2010.
28. C. Wang, J. Pang, R. Zhao, and X. Liu. Using api sequence and bayes algorithm to detect suspicious behavior. In *In Proc. of the Int. Conf. on Communication Software and Networks (ICCSN'09)*, pages 544–548. IEEE, Feb. 2009.
29. Y. Ye, T. Li, Y. Chen, and Q. Jiang. Automatic malware categorization using cluster ensemble. In *In Proc. of the 16th ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'16)*, pages 95–104. ACM, July 2010.

30. H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *In Proc. of the 14th ACM Conf. on Computer and Communications Security (CCS'07)*, pages 116–127. ACM, 2007.