ISSN 2278-3091
Volume 9, No.1.1, 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse9291.12020.pdf
https://doi.org/10.30534/ijatcse/2020/9291.12020

Feature Selection for Malware Classification and Detection: A Literature Review

Babak Bashari Rad[1]*, Mohammad Kazem Hassan Nejad[2]
[1,2] School of Computing, Asia Pacific University of Technology and Innovation, Kuala Lumpur, Malaysia
*E-mail: babak.basharirad@apu.edu.my

ABSTRACT

Computer malware, which were widely known as computer viruses, as they were the most prominent form of malware, have come a long way. With the continuous growth in the number of computer users, the threat landscape has evolved into much more than just viruses. Malware are constantly changing their functionality, with many new types being seen (such as worms, generic Trojans, backdoors, and lately Ransomware), infection vectors (such as malicious e-mail attachments), and variety, using polymorphic and metamorphic engines to propagate new variants, with an objective to broaden their target base and remain competitive. This unprecedented surge of malware has forced researchers to investigate machine learning techniques to classify and detect them. However, feature selection, one of the most im-portant steps in machine learning, as there are countless number of approaches that have been taken in the past. The goal of this paper is to review the various feature selection approaches that have been taken in the past for malware detection and classification, and discuss on their advantages and drawbacks. We have discussed on both static and dynamic analysis techniques, each of which have their own set of proper-ties and techniques that can be meaningful for feature selection.

Key words : Malware Classification, Malware Classification, Feature, Static Analysis, Dynamic Analysis.

1. INTRODUCTION

Malware are generally conceived as harmful programs, and rightfully so. However, they root to an academic history dating back to 1949 with the first theoretical work on self-reproducing automatons by John von Neumann being done, which became the backbone of computer viruses. The first variations of malware were computer viruses. Computer viruses were initially purposed to expose the vulnerabilities within an operating system without causing any deliberate harm to the user through self-replication [1]. One of the first known computer viruses was the ELK Cloner that attached itself to the Apple II operating system. This was prior to self-replicating programs being termed “computer viruses” in a work by Cohen [2]. However, over the past decades, it has gathered considerable attention to be used for malicious purposes, and it evolved into an imminent threat landscape. Malicious software, or malware as it was coined the term in 1990, is any software that causes harm to a computer, network, or user [3].

It is important to overview the process behind analyzing malware, as it is an essential step required to determine the most appropriate and informative features that differentiate different malware and benign files and techniques needed to extract them for performing malware classification [4]. Sikorski and Honig [3] state that malware analysis is the art of dissecting a malware to figure out its functionality, how to detect it, and defeat or eliminate it. It is generally broken down into static and dynamic analysis [5]. Static analysis refers to the process of examining the code and structure of a program without running it, while dynamic analysis is done by executing the program and examining its behavior [3].

In this paper, we analyze the various techniques that have been used for feature selection as part of malware detection over the past years with their associated benefits and drawbacks. The structure of the paper is as follows. In the following section, we discuss feature selection techniques that are obtained statically, with each subsection reviewing and discussing the various techniques, and lastly the benefits and drawbacks of static analysis techniques for feature selection. In section 3, we discuss feature selection techniques that are obtained dynamically and discuss the benefits and drawbacks of dynamic analysis. Lastly, we conclude the paper based on our review.

2. STATIC ANALYSIS

There are many features that are looked at using several static techniques that can be practiced for detecting malware, such as hard-coded printable strings, byte sequences, imported DLLs, library calls, and op-code n-grams, which allow all the possible execution paths to be followed [6]. Several of these features can be extracted from analyzing the PE metadata, such as imported DLLs, PE header information, entropy, file size, hard-coded strings, while others are done by examining the byte code or the decompiled/disassembled instructions, as majority of malware can be perceived as binary files. Compiled windows executables that are packed or encrypted can be initially unpacked using disassemblers, such as IDA

Pro, revealing the unfiltered static information about a binary file [7, 8].
An important concept often used in many studies are n-grams. N-grams are the consequent characters that can form a substring from a larger string with the length n [4]. For instance, "analysis" can be broken down into several 4-grams, such as "alys" "naly" "ysis" lysi" and so on. Most experiment studies use an n value between 1 and 8 with varying effectiveness [4].

2.1. Opcode Analysis

A popular approach utilized by many researches include the analysis of operational code (opcode) of a file. An opcode is a building block of machine language. It is the instruction that specifies an operation to be performed, such as data manipulation, arithmetic, logical, and program control operations [9]. Opcodes can be extracted from the assembly files once the binary executable is disassembled using a tool such as *IDA Pro*, as used by Liu et al. [5], or *NewBasic Assembler*, as used by Santos et al. [10]. Bilar [11] conducted a study which concluded that opcode frequency distributions provide a significant statistical difference between malicious and non-malicious executables, with rare opcodes improving the predictor.

Additionally, Shabtai et al. [9] state that opcode sequences provide a better representation of an executable file as opposed to byte sequences, due to the fact that they provide information about program structure, flow, and functions.
Moskovitch et al. [12] present a full methodology based on a text categorization concept utilizing n-gram opcode sequences with their results indicating that accuracy rates of over 99% can be achieved with a training set consisting of less than 15% malicious samples.

Shabtai et al. [9] have proposed an approach using n-gram opcode patterns on over 30,000 samples. They have evaluated various n-gram sizes and concluded that 2-gram opcode patterns provided the highest accuracy rate of 96% which concurred with the findings of Moskovitch et al. [12].

Santos et al. [10] propose a method of labelling malware samples to their respective malware families based on the frequency of opcode sequences. Although, this did not provide a classification for new malware, their work was extended to classify unknown samples as malicious or benign using the same technique achieving an accuracy rate of 96% [10].
Runwal et al. [13] considered a method of classifying malware based on opcode-graph similarities as opposed to n-grams achieving an accuracy rate of 96.41%.

Hu [14] has proposed and implemented a novel framework to classify unknown malware by exploiting the instruction format of x86 architecture to represent a binary file as n-grams of opcode and using a generic unpacking algorithm with dimensionality reduction algorithm to pre-process a binary file. It is remarkably scalable as it can process over 100,000 samples within 2 hours with a varying accuracy rate of 75-80%.

Recently, Kapoor and Dhavale [15] have proposed the usage of opcode features with control flow graphs (CFG) to detect malware utilizing bi-normal separation as a feature scoring metric, concluding that using bi-normal separation outperforms the traditional document frequency methods, achieving a 99.5% accuracy rate.

Liu et al. [5] have also taken the idea of using of opcode features with control flow graphs for classifying malware. However, they have also included imported function calls as a set of features for their classification, achieving a 98.9% accuracy rate.

Sharma and Sahay [16] have proposed an interesting novel approach by detecting advanced malware based on the opcode occurrences. They applied this concept to a dataset consisting of 11,088 malware samples, and 4,006 benign samples, utilizing five classification algorithms and achieved the maximum accuracy of 97.95% by using Random Forest.

2.2. String Extraction

String extraction is another popular static analysis technique used to identify malware [17]. When a program is compiled, there are certain non-encoded strings that are embedded within the binary that can be extracted and analysed [18].

For instance, Tian, et al. [19] extracted printable strings from the library code of 1367 samples achieving a 97% classification accuracy. Islam, et al. [20] extended the work by combining the printable strings extracted from a malware sample with the function lengths to perform classification achieving a 98% classification accuracy.

Recently, Shijo and Salim [18] proposed the use of Printable String Information (PSI), extracting only the useful string information that are not created by code obfuscation, coupled with behavioral information to classify malware, achieving an accuracy rate of 98.7%.

Schultz, et al. [21] proposed the first usage of static features for malware classification using three different static features, including string feature, which resulted in a classifier with double the detection rate when compared to the traditional signature-based detection methods at the time.

2.3. Byte Sequence

Another popular approach utilized by many researchers to statically analyze a file is by examining the byte content of a file. Byte n-grams are often helpful as they include information from all the PE sections as opposed to opcode n-grams [4].

Schultz et al. [21] initially utilized byte sequences as the third feature in their approach. By utilizing hexdump, the authors extracted the byte sequence n-grams and utilized a

Multi-Naïve Bayes algorithm to achieve a high classification rate of 96.88%. Although the n value has not been explicitly specified, based on an example provided in their paper, it can be deduced to be 2.

Kolter and Maloof [22] improved the results by solely utilizing n-grams of byte codes as their features, however their proposed approach was merely a pilot study to determine the size of n-grams that needed to be used, and Kolter and Maloof [22] revisited the approach proving its real-world practicality, improving the results by achieving an accuracy rate of 98%.
Tabish et al. [23] proposed a novel approach which did not memorize the specific byte sequences or strings within a binary file, claiming this will allow the non-signature-based technique to detect previously unknown and zero-day malware. By leveraging standard machine learning algorithms to classify the binary files, they achieved a maximum accuracy rate of 96.2%.

Jain and Meena [24] proposed a malware detection method based on extraction of n-grams of raw byte patterns. Their method utilized Random Forest classifier, which they experimented it with 1,018 malware samples and 1,120 benign samples, achieving an accuracy of approximately 99%, claiming that 3-gram performed better than other n-variants as the size of n-gram is inversely proportional to the frequency of the relevant n-gram.

Qi et al. [7] have taken a different approach to classify malware variants by creating a byte randomness profile based on the frequency of the varying unsigned bytes within binary files, claiming that although malware variants, which are created through self-mutation, might change the signature, the byte distribution remains the same. The approach utilizes the Cosine Similarity (COS) and Sum of Squares Distance (SSD) to measure the distinctiveness between malware samples, however with little detail on the accuracy of their model.

Recently, Singh and Khurmi [25] also proposed and implemented a method for malware clustering based on the byte frequency, as represented as time series using symbolic aggregation approximation algorithm to convert it into a symbolic representation and evaluating their model on 5,000 collected malware samples, achieving a precision rate of 92% and recall rate of 96%.

Nissim et al. [26] have introduced a novel approach utilizing active learning by combining three different machine learning algorithms over a 10-day test period using byte sequence n-grams as their features. Their experiments are promising as the accuracy rate increases from 90.05% on day one to 97.83% on the last day, however they are still less accurate than some works that have been mentioned. Furthermore, they are vulnerable to byte substitution, injection, and re-ordering.

Dinh et al. [27] have proposed an interesting approach for identifying malware from known samples. Their approach is based on sequence alignment, which is often used to identify different spices matching DNA segments, and it is implemented by using distance matrix alignment to find the longest common sequence of bytes within a malware. However, it is a time-consuming process, such as recorded by their evaluation of 51 minutes of runtime on the Apache Spark framework [27].

2.4. Other approaches

Other static analysis techniques have also been utilized by researchers over the years achieving mixed results. For instance, Gonzalez and Vazquez [28] proposed an approach utilizing the number of Application Programming Interface (API) calls that are made from the Dynamic Link Libraries (DLLs) imported by an application, since user-level programs require APIs to communicate with the operating system and utilize hardware resources. To do so, they disassembled the binary samples obtained using IDA disassembler, and used various learning algorithms to train their neural network model, achieving the maximum accuracy rate of 97.95% when utilizing Leven-Marquardt algorithm with one hidden layer for their neural network. Several researchers have also proposed techniques for malware detection revolving around statically analyzing the API calls of a sample.

For instance, Wang et al. [29] proposed an approach based on the API sequences of a malware following the de-compilation analysis using Naïve Bayes algorithm, achieving an accuracy rate of 93.71%.

Iwamoto and Wasaki [30] have also proposed a method of malware classification based on the API function calls. However, their proposed method is set to classify malware based on the presence and absence of consecutive pairs of API calls which are compared to samples from the same malware family using Dice's coefficient to determine the degree of similarity. Their work included 4,684 malware samples, from which 1,821 samples were used for the feature extraction to apply for their classification method, which was an issue within their work as they were unable to unpack certain samples and hence, the imported address tables (IATs) were corrupted.

In order to identify packed or obfuscated malware, several methods have been proposed by researchers Ugarte-Pedrero et al. [31] and Lyda and Hamrock [32] utilizing entropy analysis, which provides a measurement for the amount of uncertainty within a series of bytes that is common within compressed or ciphered data.

Saini et al. [33] have taken a different approach to propose a scalable method for malware classification using function call frequency and suspicious section count as their features. By using a dataset consisting of 2,460 malware and 627 benign samples, they have achieved maximum accuracy rate of 98.35% using J48 classifier.

Narouei et al. [34] proposed a technique to statically extract the DLL dependency tree of a portable executable. The DLL dependency tree is constructed as the portable executable is dependent on certain DLLs which have dependency on other DLLs for completion of their tasks. They implemented and

evaluated this approach with a dataset consisting of 11,000 malware samples and 4,700 benign files, achieving a maximum accuracy rate of 98.5%. However, the pitfall with their approach is that it is dependent on the Import Address Table (IAT) of the binary file, which can be removed by certain packers and requires other third-party software to be reconstructed.

Belaoued and Mazouzi [35] have proposed a novel approach to classify malware based on the static analysis of the PE Optional Header information. They experimented using a dataset consisting of 338 malware files and 214 benign files, using the chi-square method (KHI^2) for efficient feature selection and Rotation Forest classifier, achieving an accuracy rate of 97.25%. However, they do not experiment with combining different set of features which could potentially increase their accuracy rate, given that certain information from the PE optional header can be tampered.

2.5. Benefits and Drawbacks

One of the major advantages posed by static analysis is the low resource intensiveness and better performance, especially when coupled with machine learning, where training the model itself can be very resource intensive, performance demanding, and time consuming (high time complexity). Therefore, it is far more efficient than dynamic analysis [5]. Secondly, it is a safe approach, compared to dynamic analysis, as there are no chances of infecting the machine due to the fact that the malicious software is not executed and hence the binary file cannot detect the analysis process, even if it is within a virtual environment, as there have been cases of vulnerability flaws for virtual environments that certain malware have exploited in the past (Wueest), and malware authors will always be on the hunt for finding such vulnerabilities, which can be detrimental especially to a non-isolated machine. However, there is a drawback to using static features for malware classification, as the approach can be deteriorated by a malware using obfuscation or packing techniques which hide the functionality and purpose of a binary file and make it more challenging to reverse [36]. Packing refers to the act of compressing a program which can impede the analysis of the file by making the reverse engineering process more cumbersome [3]. Another way malware authors attempt to obfuscate a malware is through noise insertion. Noise insertion is the process of adding unnecessary sequences of instructions which do not change the behavior of the malware, but introduce noise, such as inserting no-operation (NOP) instructions [3].

However, as previously mentioned, there have been studies done to help identify packers and obfuscated files, and furthermore, the binary files can be disassembled and unpacked before statically analyzing them [31, 32].

3. DYNAMIC ANALYSIS

On the other hand, dynamic analysis is usually done through the execution of a malware within a controlled environment, such as a sandbox, virtual machine, or utilizing debuggers such as *OllyDBG* [37]. By executing a program within an emulated environment, the analyst can control different aspects of the program execution. It is usually done by emulating components such as memory and CPU, however it can be evaded by malware that can detect the imperfections of CPU emulation and not exhibit their actual characteristics poisoning the analysis [38].

Furthermore, dynamic analysis can be done within a virtual machine, which provides a strong isolation due to the physical machine not being directly accessible through the virtual machine environment. Virtual machines can also provide the ability to reset the analysis environment through the usage of snapshots which restore the virtual machine to a previously saved state in a short period of time [3].

3.1. Function Hooking and API Calls

Functions within an executable are often implemented to perform a specific task, such as deleting a file or renaming a file. These are often used to promote better maintainability and re-usability, however from an analysis perspective, the interesting characteristic of functions are the abstraction of implementation detail to a more semantically rich representation. For instance, there are numerous search algorithms that can be implemented that can provide the same output, however the underlying implementation can define the behavior of the program. In order to observe the functions that are called during runtime, the program functions can be intercepted by hooking. Hooking allows the functions to be monitored whenever a function is invoked by logging its invocation [1].

Typically, a function consists of code that performs a specific task, such as, calculating the factorial value of a number or creating a file. While the use of functions can result in easy code re-usability, and easier maintenance, the property that makes functions interesting for program analysis is that they are commonly used to abstract from implementation details to a semantically richer representation. For example, the particular algorithm which a sort function implements, might not be important as long as the result corresponds to the sorted input. When it comes to analyzing code, such abstractions help gaining an overview of the behavior of the program. One possibility to monitor what functions are called by a program is to intercept these calls. The process of intercepting function calls is called hooking. The analyzed program is manipulated in a way that in addition to the intended function, a so-called hook function is invoked. This hook function is responsible for implementing the required analysis functionality, such as recording its invocation to a log file, or analyze input parameters [1].

For Windows operating system, the set of functionalities that are provided by the operating system that are through API calls are packaged into Dynamic Link Libraries (DLLs). These provide different level of abstract functionalities such as network, security, file manipulation, and system services [1]. There are different system privilege levels that provide layers of security encapsulating certain functionalities. For

instance, a program running in user-mode cannot directly write or delete a file. To do so, the program needs to access the kernel mode through system call interface. For instance, if a program requires the creation of a file, it will pass the required parameters (such as name, file path, read/write attributes) by invoking the CreateFile function which then will be handled by the operating system within kernel mode and will return a file handle. Malware, just like any other programs, need to make use of these functions that can be monitored when running in user-mode. Therefore, a popular approach used when analyzing a malware dynamically is through the API calls due to the reasoning that it provides a solid behavioral information since the precise actions that are performed by an executable on a computer are shown, for example, creation/deletion/modification of files or registry keys, creation of mutexes, and network activities [39].

However, these can also be evaded by malware running directly in kernel mode [1]. A function call trace can be constructed based on the function calls with the parameters passed that have been monitored, which can provide semantically rich analysis of a binary file. This allows an analyst to compute similarities between call traces to identify malware [1].

3.2 Function Parameter Analysis

Another important feature that are analyzed through dynamic execution are the function parameters. Opposed to static analysis, dynamic function parameter analysis allows the analyst to view the actual arguments that are passed when a function is invoked. This can provide insightful information about the executable's behavior, for instance, if the file handler is returned from a CreateFile function and passed as a parameter into WriteFile function, this can provide a direct correlation between the two functions, or if an IP address is passed as a parameter into InternetConnect function to establish an ftp session with a malicious site to download the payload [1].

3.3 Related Works

By utilizing function calls and function parameter, an analyst can monitor the registry key changes, file system changes, such as creation or deletion of a file, network activity, such as establishing TCP connection with C&C server, creating mutexes, and payload injection and dumping [40].

Mohamad Fadli and Jantan [41] have proposed an approach framework for malware classification based on profile creation using malware behavior analysis, such as run-time analysis and resource monitoring. The behaviors of the samples are extracted through execution on two automated dynamic analysis frameworks, CWSandbox and Anubis, which then need to be manually customized by the authors. The malware are then classified into their respective malware families. However, this raises the issue of scalability, as the samples need to be manually customized, which is not feasible for the large number of samples that are seen every day.

Anderson et al. [42] have also proposed an approach for malware detection based on the analysis of graphs. These graphs are generated through the instruction traces after the sample is executed within a sandbox. A modified version of Ether malware analysis service to capture the instruction traces. The approach utilizes 2-grams to transition the Markov chain probabilities, which are treated as a graph. Kernel matrix is then used to construct similarity between the training instances. In order to measure the similarity, a Gaussian kernel, and a spectral kernel are used. The Gaussian. Kernel measures the local similarity between the graph edges, while the spectral kernel measures the global similarity between graphs. The kernel matrix is then used to train a support vector machine to classify the instances. This approach has a very high space and time complexity, as not only is dynamic analysis resource intensive, but the method of application is too, therefore raising issues of scalability and usage in real-world settings.

Firdausi et al. [43] have presented a proof of concept for malware detection. They have gathered 220 malicious PE samples and 250 benign PE samples. After which, the training dataset is passed into Anubis, an online dynamic analysis tool, which generates an XML report for the execution, including information such as registry modifications, file system changes, and TCP connections. The reports are then preprocessed into sparse vector models. They demonstrated the proof of concept by utilizing 5 machine learning algorithms, Naïve Bayes, J48, SVM, MLP, and kNN, achieving a maximum accuracy of 96.8%. However, the issue with their proof of concept is that the dataset was very small which raises issues of scalability for their concept.

3.4 Benefits and Drawbacks

There is a notable advantage by using dynamic analysis such as those mentioned in the notable works above, and that is that as packed malware will unpack the payload during execution, and as opposed to static analysis, there is no need for disassembling the file and manually unpacking. However, there are several major drawbacks to utilizing behavioral features for malware detection through dynamic analysis that are the pitfall of all the notable works above. Firstly, it is generally a time and resource intensive approach with huge preprocessing overhead raising issues of scalability such that the computational demands are hard to meet for a real-world scenario, especially when being used for malware classification using machine learning algorithms, which on their own are highly resource intensive with high time complexity [34]. Secondly, certain malware' are dependent upon a specific condition to be met before executing the malicious payload, such as time-delays or time/date conditions (e.g. logic bombs) or the malware is aware of its environment, given that it is being executed within a virtual machine or it is being executed for analysis purposes and will promptly change its behavior [44]. Thirdly, certain files cannot be executed, such as Dynamic Link Libraries (DLLs) even though they follow the portable executable structure and can be statically analyzed as one, and this is crucial as a large majority of sample files are often DLLs [34] or certain files

that are product of drive-by downloads are often executed with a specific parameter, and will not behave maliciously without being executed with the specific parameter [45]. Additionally, it can also be time-consuming and critical to set-up an isolated environment to execute the malware, as malware authors are often on the lookout for vulnerabilities within known frameworks that isolate the analysis machine [44]. Lastly, through dynamic analysis only a single path of execution is examined, which might not provide the most complete information about a sample [40].

4. CONCLUSION

In this paper, we critically reviewed two major approaches for feature selection for malware classification and detection, namely static and dynamic analysis, and delve deeper into each to review the various techniques and approaches that have been done in the past with their benefits and drawbacks. Feature selection based on the malware analysis can be viewed as comparing apples to oranges within the context of malware classification using any machine learning approach. Neither analysis technique is greatly superior over the other, as there are benefits and drawbacks to either method of analysis. Static analysis could address the issues of scalability and reduce performance overhead as the files do not need to be executed, while dynamic analysis could potentially improve the accuracy of the features as each file can be executed and packed files will no longer be an issue. Therefore, it is subjective to the intended aim of the malware detection that can guide the approach to be taken for feature selection. However, with the drawbacks of each technique, the two methods could perhaps be combined to provide more meaningful and accurate features, and investigation of hybrid analysis for feature selection could be taken into consideration in the future.

REFERENCES

[1] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools", *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012. https://doi.org/10.1145/2089125.2089126

[2] F. Cohen, "Computer viruses: theory and experiments", *Computers & security*, vol. 6, no. 1, pp. 22-35, 1984.

[3] M. Sikorski and A. Honig, *Practical Malware Analysis*. San Francisco, UNITED STATES: No Starch Press, 2012.

[4] E. Raff et al., "An investigation of byte n-gram features for malware classification", *Journal of Computer Virology and Hacking Techniques*, journal article pp. 1-20, 2016.

[5] L. Liu, B.-s. WANG, Y. Bo, and Q.-x. ZHONG, "Automatic malware classification and new malware detection using machine learning", *Frontiers*, vol. 1, 2016.

[6] C. Cepeda, D. L. C. Tien, and P. Ordóñez, "Feature Selection and Improving Classification Performance for Malware Detection", *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016, pp. 560-566.

[7] S. Qi, M. Xu, and N. Zheng, "A malware variant detection method based on byte randomness test", (in English), *Journal of Computers, Report*, vol. 8, 2013 2013. https://doi.org/10.4304/jcp.8.10.2469-2477

[8] B. B. Rad, M. Masrom, S. Ibrahim, S. Ibrahim, "Morphed virus family classification based on opcodes statistical feature using decision tree", *International Conference on Informatics Engineering and Information Science*, ICIEIS 2011: Informatics Engineering and Information Science, pp. 123-131.

[9] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on OpCode patterns", *Security Informatics*, vol. 1, no. 1, p. 1, 2012.

[10] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection", *Information Sciences*, vol. 231, pp. 64-82, 2013.

[11] D. Bilar, "Opcodes as predictor for malware", *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156-168, 2007.

[12] R. Moskovitcz, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, Y. Elovici, "Unknown Malcode Detection Using OPCODE Representation", *Intelligence and Security Informatics: First European Conference, EuroISI 2008*, Esbjerg, Denmark, December 3-5, 2008. Proceedings, D. Ortiz-Arroyo, H. L. Larsen, D. D. Zeng, D. Hicks, and G. Wagner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 204-215.

[13] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection", *Journal in Computer Virology*, vol. 8, no. 1, pp. 37-52, 2012. https://doi.org/10.1007/s11416-012-0160-5

[14] X. Hu, "MutantX-S: Scalable Malware Clustering Based on Static Features", 2013.

[15] A. Kapoor and S. Dhavale, "Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation", *Defence Science Journal*, vol. 66, no. 2, pp. 138-145, 2016.

[16] A. Sharma and S. K. Sahay, "An effective approach for classification of advanced malware with high accuracy", arXiv preprint arXiv:1606.06897, 2016.

[17] S. Gadhiya and K. Bhavsar, "Techniques for malware analysis", 2013.

[18] P. V. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection", *Procedia Computer Science*, vol. 46, pp. 804-811, 2015/01/01 2015.

[19] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An automated classification system based on the strings of trojan and virus families", *4th International Conference on Malicious and Unwanted Software (MALWARE)*, 2009, pp. 23-30.

[20] R. Islam, R. Tian, L. Batten, and S. Versteeg, "Classification of Malware Based on String and Function

Feature Selection", *Second Cybercrime and Trustworthy Computing Workshop*, 2010, pp. 9-17.

[21] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables", *Proceedings 2001 IEEE Symposium on Security and Privacy (S&P 2001)*, 2001, pp. 38-49.

[22] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild", *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2721-2744, 2006.

[23] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content", *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 2009, pp. 23-31: ACM. https://doi.org/10.1145/1599272.1599278

[24] S. Jain and Y. K. Meena, "Byte Level n–Gram Analysis for Malware Detection", *Computer Networks and Intelligent Computing: 5th International Conference on Information Processing, ICIP 2011*, Bangalore, India, August 5-7, 2011. Proceedings, K. R. Venugopal and L. M. Patnaik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 51-59.

[25] N. Singh and S. S. Khurmi, "ByteFreq: Malware clustering using byte frequency", *5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2016, pp. 333-337.

[26] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Novel active learning methods for enhanced PC malware detection in windows OS", *Expert Systems with Applications*, vol. 41, no. 13, pp. 5843-5857, 10/1/ 2014.

[27] A. Dinh, D. Brill, Y. Li, and W. He, "Malware Sequence Alignment", *IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016, pp. 613-617.

[28] L. E. Gonzalez and R. A. Vazquez, "Malware classification using Euclidean distance and artificial neural networks", *12th Mexican International Conference on Artificial Intelligence (MICAI)*, 2013, pp. 103-108: IEEE.

[29] C. Wang, J. Pang, R. Zhao, and X. Liu, "Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior", *International Conference on Communication Software and Networks*, 2009, pp. 544-548.

[30] K. Iwamoto and K. Wasaki, "Malware classification based on extracted API sequences using static analysis", *Proceedings of the Asian Internet Engineeering Conference*, Bangkok, Thailand, 2012.

[31] X. Ugarte-Pedrero, I. Santos, B. Sanz, C. Laorden, and P. G. Bringas, "Countering entropy measure attacks on packed software detection", *IEEE Consumer Communications and Networking Conference (CCNC)*, 2012, pp. 164-168.

[32] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware", *IEEE Security & Privacy*, vol. 5, no. 2, pp. 40-45, 2007. https://doi.org/10.1109/MSP.2007.48

[33] A. Saini, E. Gandotra, D. Bansal, and S. Sofat, "Classification of PE Files using Static Analysis", *Proceedings of the 7th International Conference on Security of Information and Networks*, Glasgow, Scotland, UK, 2014.

[34] M. Narouei, M. Ahmadi, G. Giacinto, H. Takabi, and A. Sami, "DLLMiner: structural mining for malware detection", *Sec. and Commun. Netw.*, vol. 8, no. 18, pp. 3311-3322, 2015.

[35] M. Belaoued and S. Mazouzi, "A Real-Time PE-Malware Detection System Based on CHI-Square Test and PE-File Features", *Computer Science and Its Applications: 5th IFIP TC 5 International Conference, CIIA 2015*, Saida, Algeria, May 20-21, 2015, Proceedings, A. Amine, L. Bellatreche, Z. Elberrichi, E. J. Neuhold, and R. Wrembel, Eds. Cham: Springer International Publishing, 2015, pp. 416-425.

[36] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection", *Computer security applications conference*, 2007. ACSAC 2007. Twenty-third annual, 2007, pp. 421-430: IEEE.

[37] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey", *Journal of Information Security*, vol. 2014, 2014.

[38] H. Yin and D. Song, *Automatic Malware Analysis: An Emulator Based Approach*, 1 ed. Springer Science & Business Media, 2013.

[39] P. O'kane, S. Sezer, and K. McLaughlin, "Detecting obfuscated malware using reduced opcode set and optimised runtime trace", *Security Informatics*, vol. 5, no. 1, p. 2, 2016.

[40] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and Classification of Malware Behavior", *Detection of Intrusions and Malware, and Vulnerability Assessment: 5th International Conference, DIMVA 2008*, Paris, France, July 10-11, 2008. Proceedings, D. Zamboni, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108-125.

[41] Z. Mohamad Fadli and A. Jantan, "An approach for malware behavior identification and classification", *3rd International Conference on Computer Research and Development*, 2011, vol. 1, pp. 191-194.

[42] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis", *Journal in Computer Virology*, vol. 7, no. 4, pp. 247-258, 2011.

[43] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection", *Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2010, pp. 201-203.

[44] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features", *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646-656, 2013.

[45] D. G. Llauradó, *Convolutional neural networks for malware classification*, Master's, Department of Computer Science, Universitat Politècnica de Catalunya, 2016.