Friend or Foe: Discerning Benign vs Malicious Software and Malware Family

Aaron Walker*, Tapadhir Das*, Raj Mani Shukla† and Shamik Sengupta*
*Department of Computer Science and Engineering
University of Nevada, Reno, Reno, USA
Email: awalker@unr.edu, tapadhird@nevada.unr.edu, ssengupta@unr.edu
†Department of Computer Science
University of Bristol, Bristol, UK
Email: raj.shukla@bristol.ac.uk

***Abstract*—Malware remains one of the gravest threats to cybersecurity, second only to social engineering or a lack of user security awareness. This is especially true for Windows systems in enterprise environments. As malware continues to evolve and frustrate legacy detection and prevention mechanisms, additional approaches are necessary to ensure security resilience. Machine learning offers many opportunities to better combat malware threats through the advantage of big datasets. Our research highlights how machine learning can be leveraged to identify malware threats with rapid results, enabling cybersecurity professionals to learn and adapt to these threats. The approach we present in this paper produces an efficient methodology to discern malware family and function through analysis of just the first 3,000 Windows system API function calls. We compare MLP, CNN, and SVM networks to determine the best performance in terms of accuracy and speed and find that MLP works the best with our dataset.**

***Index Terms*—Malware Detection, Malware Analysis, Malware Signature, Machine Learning**

I. Introduction

Malware, or malicious software, continues to threaten computer systems and networks ranging from the home office to corporate environments, including workstations, mobile devices, and Internet-of-Things. Malware continues to be successful because it is just as varied as the systems and users they intend to compromise. As world events such as technological innovations or a global pandemic alter the way in which we use computing devices, threats from malicious software continue to diversify, thereby continuing to frustrate cybersecurity professionals who seek to ensure the safety of the systems and networks they secure.

The threat of a malware-based compromise is particularly severe for enterprise environments which rely heavily upon Microsoft Windows systems. Malware compromises continue to rank higher in frequency for Windows systems than any other operating system [1], due in part to the prevalence of Windows systems located both in the enterprise as well as home environments. Malware-based compromises can result in a ransomware attack, remote access to the compromised system, data theft, or other forms of abuse by the malware author or remote attacker.

This research is supported by NSF Award #1739032.

Many tools currently exist to aid in the fight against malware, including anti-virus or anti-malware software. However, these tools are limited to known malware signatures. Endpoint threat detection software is an advancement in the fight against malware yet is mostly effective against known threats. Next-generation firewalls add intrusion prevention to their list of offerings while also relying on signatures and rules defined in advance of new, unknown threats. This produces a landscape of tools and methodologies which require significant investment in terms of both money and personnel to use them. As the ways in which users interact with business networks and applications change, the need for a dynamic approach to cybersecurity is not always reflected in the security tools made available, especially to those on a limited budget or time.

This has motivated us to develop a framework for an approach to not only identify malware from benign software, but also to discern malware family to provide greater threat intelligence. Both malicious and benign software perform many, many operations on a system when they are executed. Analysis of hundreds of thousands of behaviors performed on a system can be costly, time consuming, and may require advanced cybersecurity training. However, machine learning provides the opportunity to automate these tasks and produce more legible results, especially for organizations which lack a dedicated team of skilled human analysts.

Contribution: Our contributions in this paper include:

- An examination of Windows API system function calls, observed through behavioral analysis of malicious and benign software, provides machine learning models which have higher real-world accuracy due to an understanding of the differences between malicious and benign behaviors, as well as differences between malware families.
- A methodology of discerning malware and family through analysis of the first 3,000 API function calls made by a given software.

The rest of the paper is as follows: Section II presents an overview of related work in the field of malware analysis. Our proposed malware behavior analysis methodology, including the hardware and software setup, malware dataset, and analysis of the observed API calls are provided in Section III. Section

IV discusses our machine learning approach, experiments, and a comparison between our proposed approach with other prominent methods. The results of our experiments are discussed in Section V. Finally, Section VI concludes the paper.

II. Related Work

Machine learning is widely used in the field of cybersecurity and there are a number of different machine learning algorithms available for research [2], including decision trees and logistic regressions, to name but a few. Static analysis of malware involves inspection of the code at rest and has been shown to be successful in the classification of malware family [3]. This includes the examination of the register, operation codes, Portable Executable structured information and more. Dynamic analysis has proven effective in cases where static analysis would fail due to encryption or dynamic code loading [4], making this approach more attractive for the extraction of features for machine learning.

Dynamic analysis typically involves the use of a sandbox environment, such as Cuckoo Sandbox [5] which reports behaviors in terms of system API calls. Smith et al [6] demonstrated the potential for understanding malicious API calls through machine learning algorithms. The large number of Windows API calls found in malicious as well as benign software samples frustrates the process of feature selection for machine learning algorithms. One approach is to categorize the function calls based on their general function [7] [8] and then evaluate the entropy of these categorical functions based on Information Gain [9], which essentially is a measure of how much information a randomly chosen data element in a set will teach us about another randomly chosen element in a set. Heuristic N-Grams analysis also adopts the Information Gain technique and has been shown to be effective in distinguishing malware from benign software [10]. This shows that while both benign and malicious software perform many of the same Windows API calls, their relative frequencies are distinguishable.

Another approach for selecting which Windows API calls to use as features involve narrowing the scope of analyzed malware samples to model specific malware families, such as WannaCry ransomware [11]. Malware family classification can be enhanced with machine learning models, as shown in [8], however here we also see the same issues with feature extraction and definition. Malware authors are aware of the attempts of researchers and system defenders to identify malicious software and often employ anti-analysis features [12]. As a result there has been research into image processing with deep learning – in this way, machine learning has been used in malware classification based upon image processing, using an extracted local binary pattern [13]. There is no currently defined methodology for accurate, non-biased feature extraction of malicious behavior observed through dynamic analysis; therefore, it is not reasonable to assume that bias is restricted without a systematic approach for the comparison of machine learning models with differing datasets.

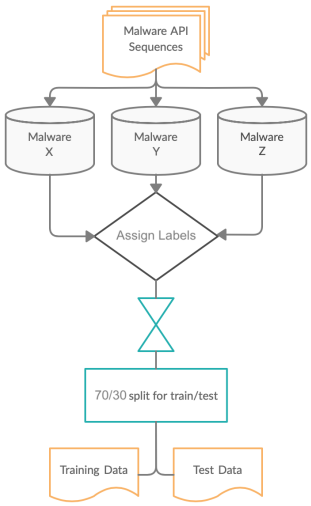

Fig. 1. Malware Data Preparation

III. Malware Behavior Analysis

In order to programmatically observe the behavior of malware in an isolated setting, we designed an environment to allow for the installation of Cuckoo and the analysis of known malware samples in a virtual machine sandbox per the installation instructions provided by Cuckoo [5]. Our goal was to focus on the evaluation of potentially malicious software affecting Windows operating systems, and the ability to discern between malware types, families, and benign software. The decision was made to focus on malware affecting Windows systems due to the significantly larger amount of malware compromises observed on Windows systems as compared to other operating systems [1], and the relative threat such malware presents to corporate environments.

A. Setup and Malware Dataset

Cuckoo was configured per the installation guide found on the Cuckoo website [5], including two 64-bit Windows 7 virtual machines installed on the Ubuntu host. Cuckoo supports many virtualization software solutions but does assume the usage of VirtualBox [14] by default, so for ease of setup we chose this platform. VirtualBox is a free system virtualization product developed by Oracle and it easily integrates with Cuckoo for administration of the virtual machines.

Known malware samples were acquired from VirusShare [15], an online resource of malicious software containing multiple versions of malware samples seen over time. This allows for the observation of evolving behaviors as the methods of exploiting system and application vulnerabilities changes with new generations of malware.

B. API Collection Methodology

Once the analysis has been performed, Cuckoo generates a report of the observed activity including, but not limited to, changes to the registry, newly spawned processes, file creation and access, virtual memory access, HTTP communication

to an external IP, and much more. When a malware file is analyzed by Cuckoo, a report is generated including a list of each behavior exhibited by the malware as it was executed in a controlled environment. This report can be delivered in JSON format, which can easily be parsed for the relevant behavior artifacts. These artifacts are expressed through the Windows API [16] system function calls made by the malware to affect the system during execution. To make use of this information, we gathered the names of the first 3,000 API function calls for each malware analyzed. We found that less than this number of API calls negatively affected the overall accuracy of our machine learning models, while a greater amount provided only little improvement to accuracy at the cost of a much greater training time. We also analyzed several commonly available software executables with Cuckoo to obtain API call behavior for comparing malicious software against benign software.

C. Malware Classification

The collection of malware available from VirusShare was delivered without any identifiers. This meant that while it was known that the supplied malware samples were malicious, there were no labels to identify the malware by type (virus, trojan, worm, etc.) or by family (Ramnit, Faceliker, Brocoiner, etc.). As we were very interested in discovering any similarities between malware types, we felt that identification of our malware samples was important. VirusTotal [19] is an online resource which allows for the comparison of the hash of each unique malware file to any malware identified by various antivirus vendors. Each of the antivirus vendors who reported that a given malware file was indeed malicious assigned a name for the family to which the malware belonged. These malware family names vary as there is no set standard nomenclature. We chose to follow the naming convention used by Microsoft antivirus, since a version of their antivirus software is available on all modern Windows operating systems, making this an appropriate baseline. As shown in Figure 1, the labeled API calls made by each malware were then collected according to their family label.

These Microsoft antivirus signatures were referenced to assign a malware family to each malware file. For the current research data set, this resulted in 23 family designations for the set of unique malware. We then labeled representative malware samples by these family names and collected the full Windows system API function calls for each malware file. For ease of reference, we assigned unique identifiers to each malware as described in Table I.

D. Benign Software API Collection

In addition to behavioral analysis of malware, we also collected Windows API calls for known benign software. This was performed to compare the behavior of benign software to malware for the purpose of showing that our machine learning methodology can discern benign activity through API analysis. The choice of what benign software to use was arbitrary, with a tendency to collect software that might be commonly downloaded by common computer users. Each of the benign software files were executed in the Cuckoo sandbox and API calls were observed in the same manner as for the malware analysis. The benign software executables as well as their reference labels are described in Table I.

TABLE I
MALWARE BY ANTIVIRUS SIGNATURE CLASSIFICATION & BENIGN SOFTWARE SET

ID	Signature	ID	Benign Executable
M1	Virus:VBS/Ramnit.gen!A	B1	7-Zip 32-bit [21]
M2	Virus:VBS/Ramnit.gen!C	B2	7-Zip 64-bit [21]
M3	PUA:Win32/Puamson.A!ml	B3	Avira Antivirus [22]
M4	TrojanClicker:JS/Faceliker.M	B4	CCleaner [23]
M5	Trojan:JS/Iframeinject	B5	Google Chrome [24]
M6	Trojan:HTML/Redirector.CF	B6	Epson scanner
M7	Trojan:Win32/Skeeyah.A!bit		software [25]
M8	Exploit:HTML/IframeRef.gen	B7	GifCam animated
M9	Virus:VBS/Ramnit.B		gif software [26]
M10	TrojanClicker:JS/Faceliker.D	B8	GIMP image
M11	TrojanClicker:JS/Faceliker.C		editor [27]
M12	Trojan:JS/Redirector.QE	B9	OpenVPN [28]
M13	Trojan:JS/BlacoleRef	B10	Ultrasurf proxy [29]
M14	PUA:Win32/Presenoker	B11	Microsoft Visual
M15	Trojan:HTML/Brocoiner.D!lib		Studio Code [30]
M16	TrojanClicker:JS/Faceliker!rfn		
M17	Trojan:Win32/Vibem.O		
M18	Trojan:HTML/Redirector.EP		
M19	Exploit:HTML/IframeRef		
M20	TrojanClicker:JS/Faceliker.A		
M21	Exploit:HTML/IframeRef.DM		
M22	Trojan:HTML/Phish		
M23	PUA:Win32/Kuaiba		

IV. MACHINE LEARNING FRAMEWORK

Our dataset of malicious and benign Windows API calls provided a description of the behavior each software performed upon the Windows system in our Cuckoo sandbox environment. Frequency analysis of these APIs for malware classification has been performed in the past [20], however we felt that the goal of our experimentation should be to discover what relationships might be observed between malicious and benign software through inspection of these API calls. To that end, we decided to make use of machine learning for analysis of API calls made by our software dataset to determine if this data was suitable for the task. We chose to use the machine learning algorithms described below to devise models which compare three sets of input data for multi-class classification. For each experiment, API calls for three software executables (malicious or benign) were used for input into these models. The APIs were input into the algorithms one at a time, so the API function names are considered in chronological order by label.

A. Machine Learning Algorithms Used

We chose to implement three machine learning algorithms for our experimentation, namely Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Support Vector Machine (SVM). These models were chosen for their applicability to classification prediction problems and accuracy performance.

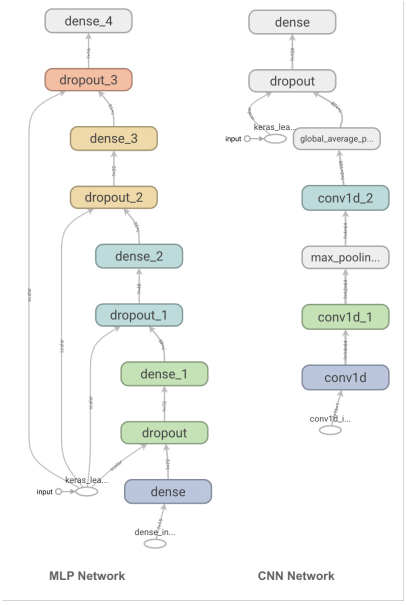

Fig. 2. MLP & CNN Network Graphs

1) Multilayer Perceptron: We built a learning network using MLP as shown in Figure 2. This consisted of five dense layers with input dropout of 40%, compiled with a categorical cross-entropy loss function and the Adam optimizer. As noted previously, we found the greatest success in our experimentation when using a network created through MLP. Therefore, the analysis in Section V reflects the results from our MLP learning model.

2) Convolutional Neural Network: We built a learning network using CNN as shown in Figure 2. This consisted of three convolutional layers with an input dropout of 50%, compiled with a categorical cross-entropy loss function and the Adam optimizer.

3) Support Vector Machine: Finally, we built a learning network using SVM. We used the linear kernel function with the one-versus-one function for multi-class classification.

4) Comparison of Algorithms: We experimented with multiple ratios of training and testing data, including 80/20, 70/30, 60/40, and 50/50, respectively. We found that in general the best performance for these machine learning algorithms for our data came with a 70/30 split between training and test data sets.

A comparison of the accuracy of these machine learning algorithms with respect to our data is shown in Figure 3. Here it is shown that the overall accuracy of the MLP algorithm performs better than CNN or SVM for each experiment. For example, experiment 1.3 resulted in an overall accuracy of 88% when MLP was used, as opposed to accuracies of 50% and 54% for CNN and SVM, respectively (more details regarding experiments and experiment numbers are provided in the next section). The only experiments where MLP did not perform better than the other algorithms were in the cases of experiment 3.3 where CNN was 2% more accurate and experiment 7.2 where both CNN and SVM were 1% more accurate. Given these small values of difference in accuracy while MLP excelled in all other experiments, we refer solely to the experimental results from usage of the MLP model in subsequent sections.

We feel it is important to try to explain why these algorithms behaved so differently. First, MLP maps the features from the input space to the output space – meaning that it takes input and then adjusts the weights during the optimization such that an optimal function is generated. This creates a more robust mapping from the input to the output so that the relationship can be learned with greater reliability when optimized. MLP can efficiently be used with a limited number of features, such as with our problem. Since our problem uses few features, MLP maps our functions from input to output spaces with the high accuracy.

CNN operates by taking convolutions of the input. CNN typically extracts a very low level of information from the input and thus helps in the classification task. The convolution and pooling layers in the CNN are typically used to extract features from image data, although they are also useful for one-dimensional data when spatial information is of interest. However, there is no spatial information in our dataset and therefore CNN probably is not performing better than MLP because of limited low-level data extraction.

SVM finds hyperplanes in a n-dimensional space, where the different classes are separated by the boundaries of the resulting planes. If the different classes are well separated, then SVM can be very effective as the defined boundaries can easily classify the dataset. It is probable that our dataset does not have well separated classes. Therefore, SVM does not work as effectively and more complex mapping between input and output using MLP performs better.

Overall, our MLP networks performed with greater accuracy than our CNN or SVM networks. As a result, we have included only the MLP results in this paper and in our appendix available online at [31].

In addition to its increased overall accuracy with our Windows API dataset, our MLP approach performed much faster than traditional CNN by a significant amount. Three days were required for training and testing each experiment using our CNN network, on average. By comparison, our MLP network required an average of four hours per experiment.

B. Experimentation

Seven experiments were devised for testing the machine learning models described in the previous section. This included the following:

1) Benign vs Trojan vs Virus
2) Benign vs Trojan vs Trojan
3) Benign vs Benign vs Benign
4) Benign vs Benign vs Malware
5) Trojan vs Virus vs PUA
6) Trojan vs Trojan vs Trojan
7) Related Malware (by signature)

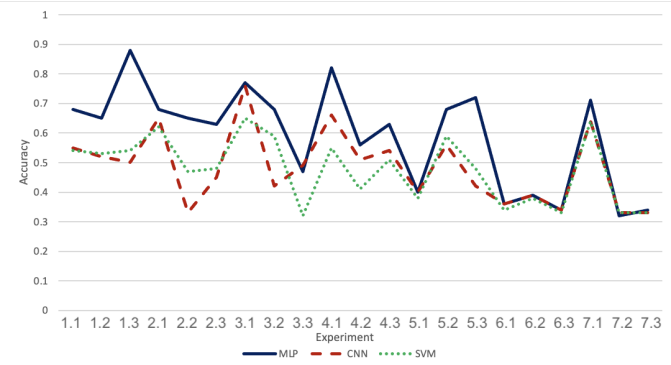

Fig. 3. Comparison of MLP Algorithm Accuracy

These experiments were purposefully chosen to assess the ability of our MLP networks to discern malware families as well as benign software from a variety of input sets of API data. Each experiment consisted of three tests, where each test concerned sets of input API calls from three different software sources. Therefore, the experiments are referred to as 1.1, 1.2, 1.3, 2.1, 2.2, 2.3, and so on through 7.1, 7.2, and 7.3.

For example, our first experiment involved the input of a benign software, a trojan, and a virus. Our third test within the first experiment is then referred to as 1.3 and involved the benign Avira Antivirus (B3), the trojan Win32/Vibem.O (M17), and the virus VBS/Ramnit.B (M9). The results from our MLP network are recorded in Table II.

Similarly, our seventh experiment involved the input of three related malwares. Test 7.1 involved APIs observed from three variants of the Ramnit virus, including VBS/Ramnit.gen!A (M1), VBS/Ramnit.gen!B (M9), and VBS/Ramnit.gen!C (M2). The results from our MLP network are recorded in Table III.

Extensive results from our experimentation can be found in our online appendix [31].

TABLE II
STATISTICS FOR EXPERIMENT 1.3

	Precision	Recall	F1-score	Support
M17	0.98	0.78	0.87	880
B3	0.83	0.86	0.84	914
M9	0.85	1.00	0.92	906
Accuracy			0.88	2700
Macro Avg.	0.89	0.88	0.88	2700
Weighted Avg.	0.89	0.88	0.88	2700

TABLE III
STATISTICS FOR EXPERIMENT 7.1

	Precision	Recall	F1-score	Support
M1	1.00	0.96	0.98	913
M9	0.55	0.74	0.63	882
M2	0.61	0.42	0.50	905
Accuracy			0.71	2700
Macro Avg.	0.72	0.71	0.70	2700
Weighted Avg.	0.72	0.71	0.70	2700

V. ANALYSIS OF RESULTS

Our initial experimentation included the experiments 5.1 - 7.3, including the classification of three sets of trojans, viruses, and potentially unwanted programs (PUA), three sets of trojans of different malware families, and three sets of related malware families. We found success in our machine learning model's ability to discriminate between different malware types, as can particularly be seen in Table II. These experiments show that there is enough of a difference in the API calls made by the malware Trojan:Win32/Vibem.O and Virus:VBS/Ramnit.B in particular to make this methodology useful for learning the type of malware being analyzed. In this way, we show that classification of malware by family is possible through observation of API function calls to learn distinct patterns.

We also saw success in the discrimination of malware of related families - in particular, our model could discern the difference between Virus:VBS/Ramnit.gen!A, Virus:VBS/Ramnit.B, and Virus:VBS/Ramnit.gen!C with high accuracy in experiment 7.1 and shown in Table III. However, we were not quite as successful in experiments 7.2 and 7.3 concerning Faceliker and iframeRef variants, respectively. We believe this is because there is more of a functional difference between the Ramnit variants, as analysis of the malware behavior shows differences in how these viruses compromise a victim computer. This suggests that our methodology is useful for determining differences in actions between malware variants of the same family and could be helpful for fingerprinting malware evolution.

Expanding our scope to include benign software in experiments 1.1 - 4.3 allowed us to determine what impact was provided by including benign software in our learning methodology. We found a much higher average accuracy for these experiments, compared to the experiments concerning only malware. We believe that the variety of benign software used in our experimentation was varied enough in function to not reflect a distinct difference in behavior from malware, as both the benign and malicious software used perform similar functions on a Windows system (file read/writes, registry changes, creation of processes to inject code, etc.). However, we found interesting results when comparing similar experiments which were differentiated by the inclusion of benign software.

For example, experiment 4.1 involved three malware inputs - namely, Virus:VBS/Ramnit.B, Trojan:JS/Redirector.QE, and PUA:Win32/Puamson.A!ml. Experiment 1.1 replaced PUA:Win32/Puamson.A!ml with the 32-bit 7-Zip executable which resulted in a change from an overall accuracy of 40% to 68%. We believe that this increase in accuracy is a result of the variance in API calls made by benign software as opposed to malware. Experiments 3.1 - 3.3 include only benign software inputs while experiments 6.1 - 6.3 include only trojan malware inputs and the former reports a greater ability to classify over the latter. This suggests that the API calls for malware lack the entropy found in benign software API calls. This would then possibly explain the greater ability to differentiate malicious

from benign software over malware of the same type.

These results show that our approach provides a means of understanding the nature of malware by learning the behavior of different malware types and family relationships. This benefits the cybersecurity incident responder by providing an additional means of malware analysis, where the relative risk presented by a certain malware not matching a current antivirus signature can be assessed by its behavioral relationship to known malware families.

VI. Conclusion

Our research showed that it was possible to discern malware and benign software through learning the Windows system API function calls made by different classifications of software. The novelty of this approach can be found in how accurately it performed, given the sparse input of single API function call names. This produced a framework for quickly learning the differences between software to accurately predict, not only if a given software is malicious or benign, but also to classify malicious software by family type. The accuracy of this approach increased when including disparate software types and we believe that the overall, general accuracy will be increased by adding additional classes with a mix of benign and malicious software. While not a replacement for current malware detection mechanisms, this approach supplied a quick tool for accurate malware analysis as part of a cybersecurity incident response process to provide greater insight and visibility into the nature of malware. One that, otherwise, may be unavailable for many cybersecurity professionals.

Acknowledgment

The authors would like to acknowledge the support of Research & Innovation and the Cyberinfrastructure Team in the Office of Information Technology at the University of Nevada, Reno for facilitation and access to the Pronghorn High-Performance Computing Cluster.

References

[1] Malwarebytes. Malwarebytes State of Malware Report 2021. Malwarebytes, February 2021. https://resources.malwarebytes.com/files/2021/02/MWB_StateOfMalwareReport2021.pdf.

[2] Liu, Qiang, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung. "A survey on security threats and defensive techniques of machine learning: A data driven view." IEEE access 6 (2018): 12103-12117.

[3] Sun, Bowen, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. "Malware family classification method based on static feature extraction." In 2017 3rd IEEE International Conference on Computer and Communications (ICCC), pp. 507-513. IEEE, 2017.

[4] Feng, Pengbin, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. "A Novel Dynamic Android Malware Detection System With Ensemble Learning." IEEE Access 6 (2018): 30996-31011.

[5] Cuckoo Foundation. "Automated Malware Analysis." Accessed July 25, 2020. http://www.cuckoosandbox.org/.

[6] Smith, Michael, Joey Ingram, Christopher Lamb, Timothy Draelos, Justin Doak, James Aimone, and Conrad James. "Dynamic Analysis of Executables to Detect and Characterize Malware." In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 16-22. IEEE, 2018.

[7] Daku, Hajredin, Pavol Zavarsky, and Yasir Malik. "Behavioral-Based Classification and Identification of Ransomware Variants Using Machine Learning." In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 1560-1564. IEEE, 2018.

[8] Pektaş, Abdurrahman, and Tankut Acarman. "Malware classification based on API calls and behaviour analysis." IET Information Security 12, no. 2 (2017): 107-117.

[9] Gandotra, Ekta, Divya Bansal, and Sanjeev Sofat. "Zero-day malware detection." In 2016 Sixth International Symposium on Embedded Computing and System Design (ISED), pp. 171-175. IEEE, 2016.

[10] Darshan, SL Shiva, MA Ajay Kumara, and C. D. Jaidhar. "Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm." In 2016 11th International Conference on Industrial and Information Systems (ICIIS), pp. 534-539. IEEE, 2016.

[11] Chen, Qian, and Robert A. Bridges. "Automated behavioral analysis of malware: A case study of wannacry ransomware." In 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 454-460. IEEE, 2017.

[12] Jain, Aruna, and Akash Kumar Singh. "Integrated Malware analysis using machine learning." In 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), pp. 1-8. IEEE, 2017.

[13] Luo, Jhu-Sin, and Dan Chia-Tien Lo. "Binary malware image classification using machine learning with local binary pattern." In 2017 IEEE International Conference on Big Data (Big Data), pp. 4664-4667. IEEE, 2017.

[14] Oracle, "Welcome to VirtualBox.org!" Accessed January 25, 2021. https://www.virtualbox.org/.

[15] VirusShare. VirusShare.com. Accessed July 25, 2020. http://virusshare.com/.

[16] Kennedy, John, Michael Satran, and Mark LeBlanc. "API Index - Windows Applications." Windows Applications — Microsoft Docs. May 30, 2018. Accessed March 23, 2019. https://docs.microsoft.com/en-us/windows/desktop/apiindex/api-index-portal.

[17] Microsoft. "TrojanDownloader:JS/Vigorf.A." Microsoft Security Int., June 30, 2016. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader%3AJS%2FVigorf.A.

[18] Reyes, Maureen. "WORM.JS.BONDAT.AC." Threat Encyclopedia - Trend Micro HK, December 13, 2018. https://www.trendmicro.com/vinfo/hk/hreat-encyclopedia/malware/worm.js.bondat.ac.

[19] VirusTotal, "Virustotal : Free online virus, malware and url scanner," https://www.virustotal.com/en/documentation/, January 2014.

[20] Walker, Aaron, and Shamik Sengupta. "Malware Family Fingerprinting Through Behavioral Analysis." In 2020 IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 1-5. IEEE, 2020.

[21] 7-Zip, "7-Zip." Accessed January 25, 2021. https://www.7-zip.org/.

[22] Avira, "Download Security Software for Windows, Mac, Android & IOS: Avira Antivirus." Accessed January 25, 2021. https://www.avira.com/.

[23] CCleaner, "Speed up & Optimize Your PC with CCleaner." Accessed January 25, 2021. https://www.ccleaner.com/.

[24] Google, "Google Chrome - Download the Fast, Secure Browser from Google." Accessed January 25, 2021. https://www.google.com/chrome/.

[25] Epson, "Epson Drivers," Accessed January 25, 2021. https://ftp.epson.com/drivers/ESU_451.exe.

[26] GifCam, "GifCam." Accessed January 25, 2021. http://blog.bahraniapps.com/.

[27] GIMP, "GIMP." Accessed January 25, 2021. https://www.gimp.org/.

[28] OpenVPN, "VPN Software Solutions & Services For Business," November 20, 2020. https://openvpn.net/.

[29] Ultrasurf, "ULTRASURF'S REACH." Accessed January 25, 2021. https://ultrasurf.us/.

[30] Microsoft, "Visual Studio Code - Code Editing. Redefined," April 14, 2016. https://code.visualstudio.com/.

[31] Aaron Walker, Tapadhir Das, Raj Mani Shukla, and Shamik Sengupta. "Friend or Foe: Discerning Benign vs Malicious Software and Malware Family." https://git.io/J3kHS. February 2021.