A Malware Variant Resistant To Traditional Analysis Techniques

Ritwik Murali*
Dept. of Computer Science & Engineering
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham, India
m_ritwik@cb.amrita.edu

Akash Ravi
Dept. of Computer Science & Engineering
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham, India
cb.en.h4cse16001@cb.students.amrita.edu

Harshit Agarwal
Dept. of Computer Science & Engineering
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham, India
cb.en.h4cse16002@cb.students.amrita.edu

2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) 978-1-7281-4142-8/20/$31.00 ©2020 IEEE 10.1109/ic-ETITE47903.2020.264

***Abstract*—In today's world, the word malware is synonymous with mysterious programs that spread havoc and sow destruction upon the computing system it infects. These malware are analyzed and understood by malware analysts who reverse engineer the program in an effort to understand it and provide appropriate identifications or signatures that enable anti-malware programs to effectively combat and resolve threats. Malware authors develop ways to circumvent or prevent this analysis of their code thus rendering preventive measures ineffective. This paper discusses existing analysis subverting techniques and how they are overcome by modern analysis techniques. Further, this paper proposes a new method to resist traditional malware analysis techniques by creating a split-personality malware variant that uses a technique known as shadow attack. The proposal is validated by creating a malware dropper and testing this dropper in controlled laboratory conditions as a part of the concept of proactive defense.**

***Index Terms*—Analysis Aware Malware, Malware, Malware Analysis, Proactive Malware Research, Split Personality Malware**

I. Introduction

One is quite aware that the computer executes decisions and processes data based on instructions given by a computer program. This computer program is just a series or sequence of specific instructions that are given by a programmer to the computer. But, even then there are specific programs that are developed by malicious programmers that fulfill their harmful intent. Such programs are what is usually referred to as malicious programs [1]. The concept of malicious programs or malware came to the forefront when Bob Thomas, experimentally created a simple self-replicating program that was intended to illustrate a mobile application [2]. The flipside of the experiment was that it also inflicted damage upon the system.

Malware today is not of one specific kind. It is multi-functional and complicated. To manage the exponential number of malware variants appearing on the internet every single day, security analysts and product vendors employ a variety of automated tools to detect, classify or analyze malicious code either in combination or as a single entity. Malware analysis is a cornerstone of defense against malicious programs that plague the computer world. The aim of the malware analyst is to understand the working of a malicious program and then suggest techniques to mitigate that threat. Malware authors usually come up with techniques to thwart malware analysis. Code obfuscation, Self-encryption, polymorphic engine, and mutation are some of the techniques that malware authors have developed to thwart analysis. But over time, malware analysts have managed to overcome this barrier. The details are discussed later in the paper.

At this juncture, it should be mentioned that malware attacks are not just targeting financial institutions, defense sectors, and occasional unsuspecting end users. According to [3], it is not completely possible for any anti-malware tool to detect the targeted and sophisticated cyber attacks of the modern age.

This paper proposes to show that existing traditional malware analysis techniques fall short when faced with techniques that follow unexpected patterns and convoluted processes. Further, due to the limited availability of ethical research on malware attack techniques, this paper also aims to share, for research purposes, a malware variant that is resistant to traditional analysis techniques and by revealing the technique, be a part of the process of converting malware analysis from being reactive to proactive.

The following sections of this paper would shed further light on this topic. Section 2 defines the term ”Malware Analysis”, and also explains the various traditional malware analysis methods and most related literature. It also discusses a particular variant of malware called Split-Personality Malware, its analysis methods, as well as an attack technique known as the Shadow attack which is used by the proposed malware variant. Section 3 provides the foundation for the proposed malware variant, discussing the architecture and its working. It includes details on how it is resistant to traditional malware analysis techniques. Section 4 details the experimental results and observations on testing the variant under controlled laboratory conditions. Section 5 concludes the paper and describes future directions that are evolved as a result of this research.

II. Background and Related Literature

A. Malware Analysis

This is the cornerstone of dissecting and understanding malware code so as to stop similar attacks. The analysis is also important when one tries to generate signatures for particular malware. The formal definition for malware analysis is stated as “the action of taking malware apart to study it” by extending the common definition of the word “analysis” [1]. This paper extends this explanation and defines malware analysis as “a method of understanding the behavior and structure of malicious code that is as complete as possible”.

Accordingly, there are various techniques for malware analysis. The traditional methods are popularly classified into ‘Static Analysis’ and ‘Dynamic Analysis’ [4]. Depending on the type of malware involved, the kind of analysis technique used determines the more optimal and valid result [5].

1) Static Analysis: As the term suggests, Static Analysis involves analyzing the malware without running it. This analysis can be on different forms of the code; namely as actual code (which is rare), on bytecode or even assembly code. If the source code is available, then static analysis tools can be used to find memory corruption flaws [6], [7]. Static malware analysis aims to uncover any potential malicious intent within a binary executable. This technique has been the major strategy for malicious intent identification for many years. The aim is to search for specific instruction patterns within the binary executable image. This instruction pattern, called the Signature, is then used to categorize the malicious behavior. Signatures are usually regular expressions that express sequences or variations of sequences of different pieces of binaries [8].

Searching for regular expressions and patterns is not the only approach for Static Analysis. Data mining and statistical principles are often used to model the code structure within a binary executable [9] [10]. The advantage of such strategies for static analysis is that it follows a more theoretical approach to make sense of the binary code. By attempting to make sense of the code by understanding its semantics, code execution is no longer a necessity.

2) Dynamic Analysis: Dynamic analysis techniques focus on observing a program’s behavior as it executes. They often monitor running applications for malicious behavior in a simulated or virtual environment. The advantage of this approach is that one does not have to imagine and figure out if the program poses a threat or not, but can observe it. There are two different approaches to dynamic malware analysis [11].

The simplest way is to take a snapshot of the complete system state and compare the results with the snapshot of the same system after the execution of the malware. This technique analyzes only the overall effect of the malware on the system thereby providing a very coarse-grained result of the analysis. It also does not take into account the changes of temporary values/threads that are in motion while the malicious code is in execution.

Monitoring an execution environment for certain specific predefined properties is a common way employed by dynamic analysis tools to track process behavior. A popular approach to this type of analysis is to execute the program under suspicion inside a virtual environment [12] [13]. Due to the extra monitoring capabilities available, the analysis tool would be able to allow the program to execute until some suspicious activity is observed. If it is a predefined activity, the task of spotting it is easy. However, in most cases, suspicious activities usually include or are interspersed with a sequence of non-suspicious activities. This implies that a malicious program might be able to perform most or all of its malicious tasks well before it is detected. In more recent times, malware that have been able to detect the presence of analysis environments are gaining prominence [14] and in such cases, the malware would avoid executing any malicious blocks of code avoid detection.

B. Recent Trends in Malware Analysis

Recent advances in machine learning and cloud computing have made heuristic analysis methods more powerful [15]. There have been a lot of studies to find optimal machine learning models that can aid in feature selection and classification [16]. Advanced techniques such as memory forensics have enabled researchers to identify features and indicators of compromise (IOC) that were previously hard to find. There have been various approaches to categorize malware variants based on machine learning and neural networks [17]. Given the increasing magnitude of new malware that need to be analyzed, security tools tend to prioritize and identify the samples that deserve deeper analysis. This process is called ”malware triage” [18]. This concept could, however, fail to detect a deeply rooted payload that might have unsuspicious headers. There has also been an increasing interest in analyzing mobile malware for platforms like Android and IoT devices [19].

C. Anti - Analysis Techniques

As the importance of malware analysis is evident, malware writers have developed various techniques to thwart analysis. Some of the most common methods are discussed below.

1) Code Obfuscation: Obfuscation is the popular technique by which malware writers evade analysts and anti-malware scanners [20]. An old, yet relevant technique is the use of a packer. It compresses and hides the malicious payload to avoid exposure during static analysis. Certain implementations can also detect tampering using integrity checking schemes. Modern tools such as IDA Pro and OllyDbg, used commonly by malware analysts, are now capable of detecting the presence of packed code.

2) Polymorphic Malware: Obfuscation did not just ensure a static change of code. More advanced obfuscation techniques include self-encrypting and polymorphic malware. A technique by which a malware executable conceals itself while maintaining its attack pathway is called polymorphism. Inspired by the dynamically changing malware concealment technique, Dr. Alan Solomon coined the term for this functionally. The malware spreads from file to file, changing/encrypting itself along the way. In a well defined polymorphic virus, there is practically nothing common in the decryptor bytes between multiple versions, thereby ensuring that there is no pattern to match [21], [22].

D. Split - Personality Malware

As discussed previously, malware writers go to extreme extents to keep their code hidden from malware analysts. Polymorphic malware were slowly caught by analysts by making use of emulators, virtual environments, sandboxes and even through algorithmic analysis of their code [21]. This led to the development of various evasion techniques by malware authors whose aim was to thwart any means of analyzing the malware code, be it debugging, disassembly or analysis in a virtual environment. When the malware detects that the system is under analysis, it hides the malicious functionality or usually terminates without performing the malicious activity. Split-Personality malware is a variety of computer malware that can understand if it is being analyzed and change its behavior accordingly. VM detection techniques like Hardware Fingerprinting, Registry Check, Memory Check, VM Communication Channel Check, Process & File Check, and Timing Analysis are some of the techniques that stand instrumental in creating a Split-Personality malware [23].

1) Analyzing Split-Personality Malware: Automated malware analysis tools such as Anubis, typically follow a dynamic approach [24]. Since Dynamic analysis based approaches and traditional malware analysis techniques are used, the tools only observe a single execution path. Modern malware are aware of this strategy and exploit this by either making use of behaviors that make use of triggers to execute or by identifying analyzers so that malicious behavior could be hidden. This has given rise to a whole variety of attack strategies such as logic bombs (e.g. Jokra trojan), time bombs (e.g. Code Red worm), command and control bots (e.g. Zeus bot), etc. This also exposed the main problems of dynamic analysis tools, namely:

1) Limited test coverage.
2) Malware that can identify and evade the malware analysis environment.

For the first case, the problem of limited test coverage, the obvious extension is to explore multiple execution paths so as to identify the trigger that enables the program to exhibit malicious behavior. [25] had developed a dynamic analysis based technique that ventures into multiple execution paths. This is performed by storing and recursively restoring prior program states that have been saved and solving constructed path constraints. This strategy can be used to discover a concrete set of in-memory variable values that satisfy the conditions corresponding to different paths. They then perform dynamic taint analysis on the inputs gathered from system calls and construct linear constraints depicting the various dependencies within memory variables.

To address the second problem of malware that seem to be able to detect that they are being observed and act accordingly, researchers have proposed stealthy (transparent) analysis tools [26] and rules [27] that are more difficult to identify. These tools gather system call traces in a more effective and efficient manner. However, in case a more fine-grained analysis is required, especially one that includes more than system calls, the tools have to resort to a model in which each of the individual instructions is inspected and logged. Unfortunately, for a more complete and comprehensive analysis, all automated tools such as Anubis, need to see more than a system call trace. For example, Anubis analyses Windows API library calls, and it also tracks data flow dependencies [24].

E. The Shadow Attack

As behavior-based malware detection and analysis gathered popularity, the attackers have also developed similar techniques to evade these behavior-based malware detection engines. The main concept off the Shadow Attack involved partitioning one piece of malware into multiple “shadow processes” [28]. None of these shadow processes contain a behavior recognized by a single process based malware detectors as malicious. In most cases, behavior-based malware detectors make use of system calls or sequences of system calls/ graphs that make use of simple inheritance or branches of individual processes of single process programs to identify the malicious characteristics. The shadow attack technique exploits this characteristic of the analysis program. Since the malware analysis program looks for a sequence of system calls from a single process in order to mark as malicious, single system calls in the same sequence that is performed by multiple shadow calls are not detected. Further, as behavior-based malware detection has become more and more prevalent, the feasibility of hiding explicit Shadow Process Calls (SPC) by mixing in multiple implicit chains was analyzed. This includes chains that occur using remote network connections to coordinate the attack. Dynamic information flow and data tainting based detection are commonly used in dynamic analysis. Both of these techniques assume that assembly instructions are mainly data-dependent. This assumption is completely violated as the shadow process hides the local SPCs by converting the process from data dependence to control dependence [29].

Further, a compiler-level prototype tool named AutoShadow has been developed for C/C++ based malware codes. It is meant for malware writers to make the source to binary and source to source conversions automatically [28]. The author also claims to have applied AutoShadow on real-world malware successfully to show that behavioral detection/analysis tools can be evaded by the shadow process. This paper assumes that the obtained results are accurate and makes no attempt to test this functionality of AutoShadow. This paper also makes use of a proposed variant of AutoShadow attack technique for a proof of concept implementation of the proposed malware variant.

III. The Proposed Variant

Symantec Corporation states that a malware dropper is a means to an end rather than the end itself. This is because a dropper itself is not an attack but the initial stage of the attack. According to Symantec, droppers primarily act as containers so as to transfer a malware payload from a source computer to another destination. The execution of a dropper implies that it just loads itself into memory and writes its malware payload into the target filesystem. In some cases where the malware payload has to be installed, the dropper also performs the installation procedures. Once the malware has been either written into filesystem or installed, as required by the payload, the task of the dropper is complete and it stops all activities. Droppers are usually used by most malware creators to hide their malware.

It has been increasingly proven that insider attacks are a major concern for most multinational corporations [30]. Threats like sabotage, espionage, and unauthorized trading are on the rise due to practices like 'bring your own device' (BYOD). According to EY's Global Information Security Survey in 2015, 56% of the participants consider their own employees to be the second most likely source of a cyberattack. This malware variant also makes use of this idea to target the victim from within the same network

The proposed variant is a split-personality malware that combines the concept of analysis aware malware and the shadow attack technique. The objective was to develop a malware dropper that could not be analyzed using traditional malware analysis techniques. The research implements a malware dropper and specifically prevents analysis by debugging tools.

A. Architecture

The proposed malware is a variant that attacks other computers by making use of their network. The variant usually attacks nodes from within the same network. It makes use of a simple peer exchange to escape detection. The list of notations used in the proposed attack model is shown in Table I.

TABLE I
Notations used in the proposed attack model

Description	**Set**	**Instance**
Set of Networks	$N = (n_1, n_2, \dots, n_u)$	$\forall i \in [1,.,u], n_i \in N$
		$\forall i \in [1,.,v],$
		$\forall j \in [1,.,u], c_i$
Set of Computers	$C = (c_1, c_2, \dots, c_v)$	$\in C \wedge C \subseteq n_i$
Set of Process	$P = (p_1, p_2, \dots, p_w)$	$p_i \in P \forall i \in [1,.,w]$
Set of Process States	$Q = (q_1, q_2, \dots, q_x)$	$q_i \in Q \forall i \in [1,.,x]$
Set of System Calls	$S = (s_1, s_2, \dots, s_y)$	$s_i \in S \forall i \in [1,.,y]$

Based on the notations shown in Table I, the attack model of the malware variant which starts from a location $c_1 \subseteq n_1$ is elaborated in equations (1) (2) and (3).

$$detect(debugger) = \begin{cases} True, & \text{if any ActiveDebugger found} \\ False, & \text{otherwise} \end{cases} \tag{1}$$

$$malware(attack) = \begin{cases} don't\_execute\_malcode, & \text{if detect(debugger)} \\ perform_{attack}, & \text{otherwise} \end{cases} \tag{2}$$

$$perform_{attack} = \begin{cases} \big[\ c_1\ (s_1 \rightarrow s_2) \rightarrow c_2\ (\Delta s_2) \rightarrow c_3\ (s_3 \rightarrow s_4) \\ \rightarrow c_1\ (\Delta s_4)\ \big] \subseteq n_1 \end{cases} \tag{3}$$

Figure 1 describes the attack module of the malware variant. From the definitions, it is evident that c1, c2, and c3 are three different computers that are present within the same network. p1, p2, and p3 are three processes that are a part of the proposed variant. p1 performs the split personality aspect of the malware, p2, and p3 take care of the shadow attack concept.

The system calls available to AutoShadow are classified into two broad function categories [28].

1) File I/O Operations
   a) File Open
   b) File Read
   c) File Write
2) Network Operations
   a) Socket
   b) Connect

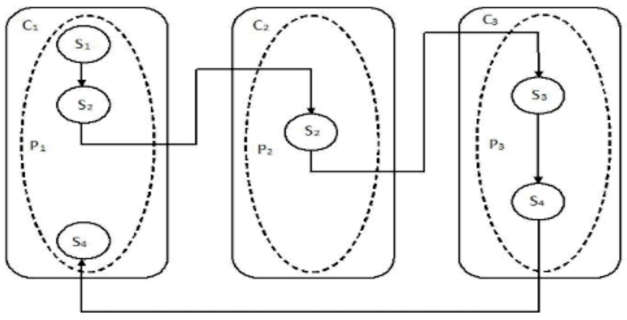

Fig. 1. Attack Architecture of the malware variant

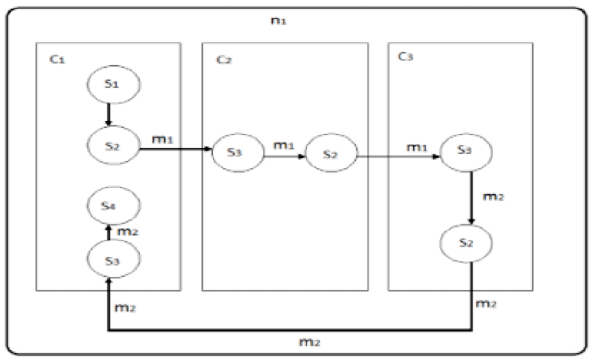

Fig. 2. A Simple Attack Scenario

c) Receive
d) Send
e) Read
f) Write

Based on these calls available, the proposed variant makes use of two network operations (sendMessage(), receiveMessage()) and one file operation (writeFile()) to fulfill the shadow attack part. The Split-Personality aspect of the malware is handled by the operation debugDetect(). Therefore in Figure 1, the symbols s1, s2, s3, and s4 correspond to debugDetect(), sendMessage(), receiveMessage(), and writeFile() respectively. Messages m1 and m2 are shared among the various processes where m1 is the signal of the dropper to be active and m2 is the malware that is being dropped by the proposed malware dropper.

As mentioned, the malware variant combines the concept of a split-personality malware with the shadow attack. Since the target OS was Microsoft, the malware was developed using C#. The language also allows one to directly import the various Microsoft Windows DLL files and built-in operations without packing it along with the malicious code. The kernel32.dll is one such Windows DLL that is used by the malware. This is because using the functions IsDebuggerPresent() and CheckRemoteDebuggerPresent() present within the DLL is easier than embedding the same functionality within the malware.

B. Working of the Malware Variant

The primary assumption is the fact that the malware has infected a computer c1 within a network n1. For the purpose of simulation, the variant carries the old but famous Shortcut Virus as its payload. This is nothing but a simple .js file that super hides all folders in any location and displays only the shortcut to the location to the user. Each part of the three-part variant contains a detection engine that performs the role of a split-personality malware by detecting the presence of debuggers. Since this was a proof of concept malware variant, it was not made to be an autostart executable, though making it auto-startable is also not too much trouble. Each executable was manually started for testing purposes. In the attack scenario depicted in figure 2, 'Computer A' acts as the victim.

Computer A started Program A, one of the three variant parts. Program A announces that Computer A is a victim. The split-personality module of program A (system Call s1) executes. Only if there is no active debugger within the system the attack module executes. Program A then identifies a particular location for the payload destination and an active port for listening and then transmits this information to 'Program B', the second part of the three-part variant. Program B present in 'Computer B' is also within the same network and accepts this data and retransmits this to 'Program C' in 'Computer C'. This then transmits the payload to Computer A using the information given to it by Program B. This is clearly explained diagrammatically in Figure 2.

IV. Observations on the Variant

Before moving on to describing the observations, let it be noted that the proposed malware variant is a proof of concept dropper that specifically detects the presence of debuggers. Based on this, the following observations have been made.

The variant effectively behaves like a malware dropper and successfully delivers the malicious payload to the required host. It can be seen that on attempting to debug the bytecode for analysis, the malware variant acts in a benign manner which is its primary objective. Also, since the location and communication ports are randomized, the reverse trace of the dropped malware to back to our program is avoided. To verify that this dropper was not detected and marked as malicious by antivirus scanners, it was submitted to online malware scanners. A quick scan and Virustotal revealed that this malware variant was given a clean bill of health. Further, the only detail that tools like Anubis were able to infer was the list of DLLs accessed by the variant. Comodo Virus Scanner, Norton Power Eraser, and Malwarebytes also revealed that the malicious nature of the variant remains undetected.

A. Detection Mechanism

This version of the malware variant is vulnerable to static analysis of the PE bytecode. Using the static analysis techniques mentioned by [14], one can identify the split-personality engine of the malware. However, since the static analysis of bytecode is a tedious process, the variant accepts the vulnerability as an acceptable risk. As of now, the presence of the malware can be seen using various other techniques such as monitoring real-time process activity, logging system calls and tracing system modifications. This is because the split-personality angle of the proposed malware focused only on debuggers.

The malware is also vulnerable to network-based tracing as the messages m1 and m2 shared are visible over the network. However, as in the case of static analysis, since analyzing the huge amount of traffic over the network is tedious, the vulnerability remains negligible.

V. Conclusion and Possible Directions

Computer malware is definitely the dark side of computer programming. A single program can behave in such a malicious manner that it can destroy in seconds what people have taken years to achieve. Malware writing is now a very lucrative business that an entire underground economy is thriving. Ransomware attacks and Cryptomining malware have become a prominent threat to individuals and enterprises alike. Due to the arrival of such variants, supporters of the anti-malware industry have taken the stance of "proactive defense". This implies that academicians and researchers are encouraged to identify novel means of creating malware so that they can be effectively blocked before causing widespread mayhem.

This paper has explored a variant of the split personality malware that is resistant to traditional malware analysis techniques. Since the variant, is itself a three-part malware, that requires intercommunication between the parts, traditional analysis techniques do not apply to it. Analyzing network communication from a completely non-infected system seems to be the only foolproof method to identify the presence of this malware variant. This, of course, implies that malware analysis needs to move from a host-based analysis to a more network-based approach. Fortunately, the Security information and event management (SIEM) systems used in enterprises generally monitor the network for malicious activities and use a trust model to evaluate their threat. The difficulty here is that network traffic analysis along with malware analysis is too tedious due to the large volume of data. One might argue that automated network analysis tools can be used to scan for the "PUSH" messages through the network. But to the malware writer, hiding data within the packet headers is no new task. Further, by scanning the headers of each and every packet, the network administrator would be forced to choose between network speed and network security.

This malware variant is only a proof of concept model. However, even analysis of this is a challenge for malware analysts. A few other possible variations of this malware variant such as the inclusion of a self-destruct or metamorphic engine also adds to the difficulty of malware analysis. As malware writers evolve, this type of malware is definitely going to be a huge bane for security researchers and anti-malware supporters. This variety of malware forces the network administrators to even rethink their original access control policies. Future research in malware analysis needs to focus on this type of malware variants and its analysis methods.

References

[1] K. Kendall and C. McMillan, "Practical malware analysis," in *Black Hat Conference, USA*, p. 10, 2007.

[2] P. Szor, *The Art of Computer Virus Research and Defense: ART COMP VIRUS RES DEFENSE _p1*. Pearson Education, 2005.

[3] V. Benson, J. McAlaney, and L. A. Frumkin, "Emerging threats for the human element and countermeasures in current cyber security landscape," in *Cyber Law, Privacy, and Security: Concepts, Methodologies, Tools, and Applications*, pp. 1264–1269, IGI Global, 2019.

[4] R. Sihwail, K. Omar, and K. A. Zainol Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," vol. 8, pp. 1662–1671, 11 2018.

[5] S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi, "A survey on malware analysis and mitigation techniques," *Computer Science Review*, vol. 32, pp. 1–23, 2019.

[6] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[7] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.

[8] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand, "Automated synthesis of semantic malware signatures using maximum satisfiability," *arXiv preprint arXiv:1608.06254*, 2016.

[9] P. Khodamoradi, M. Fazlali, F. Mardukhi, and M. Nosrati, "Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms," in *2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, pp. 1–6, IEEE, 2015.

[10] L. Xiaofeng, Z. Xiao, J. Fangshuo, Y. Shengwei, and S. Jing, "Assca: Api based sequence and statistics features combined malware detection architecture," *Procedia Computer Science*, vol. 129, pp. 248–256, 2018.

[11] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 3, 2018.

[12] C. A. Benninger, *Maitland: analysis of packed and encrypted malware via paravirtualization extensions*. PhD thesis, 2012.

[13] M. Ritwik and K. Praveen, "Analyzing the makier virus," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 2 Part 1, p. 530, 2013.

[14] R. R. Branco, G. N. Barbosa, and P. D. Neto, "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies," *Black Hat*, 2012.

[15] P. Najafi, A. Mühle, W. Pünter, F. Cheng, and C. Meinel, "Malrank: a measure of maliciousness in siem-based knowledge graphs," in *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 417–429, ACM, 2019.

[16] R. M. R. G. KannanMani ManiArasuSekar, Paveethran Swaminathan and S. Surya, “Optimal feature selection for non-network malware classification,” IEEE, 2020. Accepted for publication in the 5th International Conference on Inventive Computation Technologies (ICICT-2020).

[17] D. Ucci, L. Aniello, and R. Baldoni, “Survey of machine learning techniques for malware analysis,” *Computers & Security*, 2018.

[18] G. Laurenza, R. Lazzeretti, and L. Mazzotti, “Malware triage for early identification of advanced persistent threat activities,” *arXiv preprint arXiv:1810.07321*, 2018.

[19] H. Alasmary, A. Anwar, J. Park, J. Choi, D. Nyang, and A. Mohaisen, “Graph-based comparison of iot and android malware,” in *International Conference on Computational Social Networks*, pp. 259–272, Springer, 2018.

[20] J. Singh and J. Singh, “Challenges of malware analysis: Obfuscation techniques,” *International Journal of Information Security Science*, vol. 7, p. 100, 2018.

[21] N. S. Selamat, F. H. Mohd Ali, and N. A. Abu Othman, “Polymorphic malware detection,” in *2016 6th International Conference on IT Convergence and Security (ICITCS)*, pp. 1–5, Sept. 2016.

[22] A. Miraglia, “Analysing the spreading of computer worms and viruses: potentials and limits,” *Department of Computer Science, University of Zurich*, 2011.

[23] K. Vishnani, A. R. Pais, and R. Mohandas, “Detecting & defeating split personality malware,” in *The Fifth International Conference on Emerging Security Information, Systems and Technologies*, pp. 7–13, IEEE Computer Security, 2011.

[24] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, “A lustrum of malware network communication: Evolution and insights,” in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 788–804, IEEE, 2017.

[25] R. Sahita, X. Li, L. Lu, L. Deng, A. Shepsen, X. Xu, L. Huang, H. Liu, and K. Huang, “Executing full logical paths for malware detection,” June 29 2017. US Patent App. 14/998,178.

[26] G. Pék, “New methods for detecting malware infections and new attacks against hardware virtualization,” 2015.

[27] S. Naveen and T. G. Kumar, “Ransomware analysis using reverse engineering,” in *International Conference on Advances in Computing and Data Sciences*, pp. 185–194, Springer, 2019.

[28] W. Ma, P. Duan, S. Liu, G. Gu, and J.-C. Liu, “Shadow attacks: automatically evading system-call-behavior based malware detection,” *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 1–13, 2012.

[29] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao, “Impeding behavior-based malware analysis via replacement attacks to malware specifications,” *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 3, pp. 193–207, 2017.

[30] W. Li, W. Meng, L.-F. Kwok, and H. Horace, “Enhancing collaborative intrusion detection networks against insider attacks using supervised intrusion sensitivity-based trust management model,” *Journal of Network and Computer Applications*, vol. 77, pp. 135–145, 2017.