University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Virtual File System for Hostbased Moving Target Defence in IoT Devices

Rinor Sefa
Prizren, Kosovo
Student ID: 19-771-369

Supervisor: Jan von der Assen, Dr. Alberto Huertas Celdran
Date of Submission: October 11, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Cryptographic ransomware encrypts files and demands a ransom for their decryption. Ransomware is increasingly targeting Internet of Things (IoT) devices that contain critical data. Due to limited resources, IoT devices cannot implement resource-intensive protection mechanisms to defend against ransomware. To provide a lightweight ransomware protection mechanism for IoT devices, three overlay file systems have been implemented. The overlay file systems use moving-target defence techniques to hide file type identification, increase encryption time, and trap ransomware in infinite directories. The evaluation results show that the implemented overlay file systems provide protection against ransomware attacks. The main limitations of the overlay file systems are the inability to distinguish between malicious and non-malicious applications and the performance overhead for small file sizes.

Abstract Deutsch

Kryptografische Ransomware verschlüsselt Dateien und fordert Lösegeld für ihre Entschlüsselung. Ransomware zielt zunehmend auf Internet of Things (IoT)-Geräte ab, die kritische Daten enthalten. Aufgrund begrenzter Ressourcen können IoT-Geräte keine ressourcenintensiven Schutzmechanismen zur Abwehr von Ransomware implementieren. Um einen einfachen Ransomware-Schutzmechanismus für IoT-Geräte bereitzustellen, wurden drei Overlay-Dateisysteme implementiert. Die Overlay-Dateisysteme verwenden Moving-Target-Verteidigungstechniken, um die Dateitypidentifikation zu verbergen, die Verschlüsselungszeit zu erhöhen und Ransomware in unendlichen Verzeichnissen einzufangen. Die Bewertungsergebnisse zeigen, dass die implementierten Overlay-Dateisysteme Schutz vor Ransomware-Angriffen bieten. Die Haupteinschränkungen der Overlay-Dateisysteme sind die Unfähigkeit, zwischen schädlichen und nicht schädlichen Anwendungen zu unterscheiden, und der Leistungsaufwand für kleine Dateien.

Acknowledgments

I'd like to thank Jan and Alberto, my thesis advisors, for their guidance. I'd like to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to complete my master's thesis at the Communication Systems Group. I'd like to thank the GoFuse project's creators and maintainers for creating and maintaining such a wonderful project. I'd like to thank the developers of IOzone for creating the tool for evaluating file system performance. I'd like to thank the authors of the Govdocs1 corpus for their hard work in compiling the data set. I'd like to thank every author whose work I have referenced. I'd like to express gratitude to all the lecturers who contributed to my education. I'd like to thank my family and friends for their continued support. Finally, I would like to thank the family of my uncle Ferid for supporting my stay in Switzerland.

Contents

Chapter 1

Introduction

Ransomware is defined as ”malware that inhibits the use of resources until a ransom, usually monetary, is paid” [1]. Cryptographic ransomware is a type of ransomware that encrypts files and then demands payment to decrypt them. Ransomware targets users of all types, including end users, governments, and hospitals, on a range of platforms including PCs, workstations, mobile devices, and IoT devices [2]. In press releases [3], [4], the U.S. Department of Treasury reported that $590 million in ransomware payments were reported in the U.S. in the first half of 2021, compared to a total of $480 million in 2020, which was more than four times the amount in 2019. While ransomware payments can be measured, the damages are unmeasurable, as some files lost can not be traded for monetary value for users. Although research on ransomware defences has advanced, protection is still necessary because ransomware are continually evolving.

To accomplish ransomware's goal of encrypting files, cryptographic ransomware interacts with the file system. The file system is responsible for storing, retrieving, locating, and manipulating files. This work aims to defend against cryptographic ransomware by leveraging the file system and the cyber defence paradigm known as Moving Target Defence. Moving Target Defence is a cyber defence paradigm that was introduced in 2009 [5]; it aims to increase the effort required to exploit a vulnerability by changing system attributes, thereby making the attack surface unpredictable.

To protect against ransomware, three overlay file systems utilising moving target defence techniques were implemented. The file type file system protects against ransomware that targets specific file types. The file type file system removes elements that indicate the type of file, such as the file's name extension and file's signature bytes.

The infinite-directory file system is implemented for ransomware that traverses the file system using a depth-first strategy. The ransomware is trapped in an infinite directory tree. The files are not encrypted because encryption in a depth first strategy begins after traversal.

The slow-read-write file system reduces the read and write speeds of encryption. It reduces the speed by modifying the read and write system call parameters and by minimizing file system performance enhancements.

The evaluation result demonstrates that the file-type file system is effective against RansomPoc and DarkRadiation ransomware, as these ransomware use file name extensions to determine whether a file should be encrypted. The infinite-directory achieves its objective against RansomPoc and DarkRadiation, as the ransomware were unable to encrypt files after becoming trapped in the infinite directory tree. The slow-read-write file system also achieves its goal of increasing the encryption time for DarkRadiation from 1 minute and 14 seconds to 15 minutes and 7 seconds. Even though the purpose of the slow-read-write file system is not to protect against encryption, the RansomPoc encrypted files were recoverable due to the features implemented in the slow-read-write file system.

The main limitation of the implemented file systems is that they cannot distinguish between malicious and non-malicious applications. Consequently, malicious applications can also be impacted by the file system's features. In addition, the implemented file systems lack the read and write speeds of the default file system. For reading small files, the performance of the default file system is up to 2.9 times greater than that of the implemented file systems and up to 2.1 times faster for writing small files.

1.1 Thesis Outline

Chapter 2 introduces the topics of moving target defence, malware, file systems, and the electrosense project. Chapter 3 describes previous work on moving target defence techniques and ransomware-aware file systems. The requirements, design specifications, and implementations of the built file systems are documented in Chapter 4. In Chapter 5, the implemented file systems are evaluated against ransomware and their performance is compared to the performance of the default file system. Chapter 6 discusses the limitations of the implemented file systems and the report.

Chapter 2

Background

The objective of the background chapter is to introduce readers to the topics covered in this report. Topics covered include moving target defence, malware, file systems, and the Electrosense project.

2.1 Moving Target Defence

Moving Target Defence (MTD) is a cyber defence paradigm first described in the 2009 National Cyber Leap Year Summit report [5] by the Networking and Information Technology Research and Development (NITRD). The MTD aims to increase the work effort needed to exploit a vulnerability. This aim is to be achieved by changing system attributes, effectively making the attack surface unpredictable.

2.1.1 Cyberattack Effort Measurement

MTD aims to increase the amount of work required to exploit a vulnerability which can be determined by analysing the flow of the attack. The cyber kill chain is one model that describes the flow of a cyberattack. The cyber kill chain is presented through a seven-layer model in the paper [6].

In the **reconnaissance phase**, the attacker gathers information about the potential target. This phase can be broken down into stages of identifying, selecting, and profiling the target. The goal of this phase is to use the gathered information to design and deliver a malicious payload. In the **weaponization phase**, the attacker designs a backdoor and a penetration plan to successfully deliver the backdoor. In the **delivery phase**, the attacker tries to transfer the malware to the target environment. In the **exploitation phase**, the attacker attempts to exploit the malware. In the **installation phase**, the attacker tries to install the malicious payload. In the **command-and-control** phase, the attacker gains remote access to the compromised machine. In the **actions and objectives phase**, the attacker takes action to achieve their goals.

Examining the cyber kill chain with the objective of identifying measures that can be used to determine the amount of work required to exploit a vulnerability is a challenging task. It is a challenging task because it is not known what is happening on the attacker's side. Details such as their level of knowledge, or available resources are not known on the defender's side. Once the attack begins on the defender's side, we can start identifying measures to determine the effort required to exploit a vulnerability. Depending on what we are attempting to defend, these measures will vary. In the evaluation section, the measures for this project are defined.

2.1.2 System Attributes and Change Techniques

MTD aims to achieve its goals of increasing the effort of attacking a system by changing system attributes. The paper [7] categorises system attributes according to the software stack model. In addition, techniques for dynamically changing these system attributes within an MTD system are proposed.

The **data layer** can be changed using dynamic data techniques. Dynamic data techniques dynamically change the data format, syntax, encoding, or representation of application data. The **application layer** can be changed using dynamic software application techniques. Dynamic software application techniques change the application code by modifying program instructions. The **runtime environment layer** can be changed using dynamic runtime environment techniques. Dynamic runtime environment techniques change the environment presented to an application by the operating system. These techniques include address space randomization, which dynamically changes the layout of memory. Instruction set randomization dynamically changes the interface presented to an application by the operating system. The **operating system layer** and part of the hardware layer can be changed with dynamic platform techniques. These techniques change the platform properties (e.g., CPU, operating system). The **network layer** is changed by the dynamic network techniques. This technique includes protocol and address changes.

Changes in system attributes can occur at a fixed period of time, at random, or in response to an event [8].

2.2 Malware

Malware is any software that intentionally causes harm. In order to understand how to defend against malware, the report describes what malware does and how it does it.

2.2.1 Malware Behavior

Based on malware behavior, [9] categorises malicious malware behaviors as:

The behavior of stealing information which compromises users confidentiality. Stolen information can be used in a variety of ways, such as to access other resources or to cause harm. Examples of information include passwords, login credentials, private keys, banking information, and private chats.

The behavior of creating a vulnerability in security is used to gain access to the victim's resources. As a result, the integrity of the system is compromised. The vulnerability can be made in numerous ways, such as when the antivirus protection is turned off, the software is downgraded, the firewall settings are changed, etc.

The availability of a service is affected by a denial of service behavior. This prevents legitimate users from using the service. Examples of denial of service are denial of access (DDos), denial of access attacks, hardware corruption, etc.

The execution behavior of CandC commands. Authors can gain complete control of a system using the CandC malware. This action jeopardizes the system's integrity. RATs are one example of this type of malware.

Deception of user behavior. This behavior compromises the system's integrity and confidentiality. Phishing and man-in-the-middle attacks are examples of this type of behavior.

Annoying user behavior. This behavior disrupts the normal workflow of the user but does not compromise the integrity, availability, or confidentiality of the system. Adware, unwanted pop-ups, etc., are examples of such malware.

Stealing computer resources behavior. This behavior is intended to generate profits. This behavior compromises the system's availability and integrity. Cryptominer malware is an illustration.

Malware designed to cause malicious behavior also causes other behaviors that aid it in achieving its objective. The paper [9] classifies these behaviors as follows: **Evading analysis** in order to remain stealthy. Techniques of evasion include privilege escalation, logic bombs, etc. **Avoiding detection** via code obfuscation, fileless malware, network evasion techniques, etc. **Supporting operations-related behaviors**, such as communicating with the CandC and gaining persistence, among others.

2.2.2 Malware Analysis

The knowledge of how malware causes damage can aid in the development of anti-malware defences. Two types of malware analysis exist: statistical analysis and dynamic analysis.

Static analysis is the process of analysing malware without running it. Techniques such as string signature, byte-sequence n-grams, syntactic library calls, control flow graphs, and opcode frequency distribution are utilised [10]. **Dynamic analysis** is the process of analysing malware as it is being executed. Dynamic analysis techniques include function call monitoring, function parameter analysis, information flow tracking, and instruction traces [10].

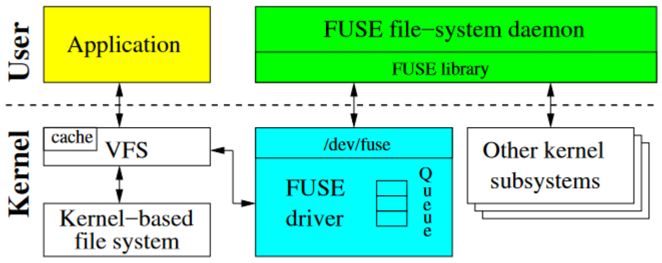

Figure 2.1: FUSE High-level Architecture [12]

2.3 File System

The file system is a component of the operating system. The file system's responsibility is to store, retrieve, locate, and manipulate files. Due to the varying task requirements, different file systems perform these tasks using different implementations.

2.3.1 Building a File System

File systems are implemented within the kernel and the user space of an operating system. Due to the complexity of the kernel, the development of file systems seems to be a challenging task. Such challenges include the complex kernel code and its data structure, a lack of development tools, kernel code debugging, and error handling [11]. User-space code is easier to develop, port, and maintain than kernel code [11]. In some cases, user space-based file systems perform worse than kernel based file systems. When a workload contains numerous metadata operations, the performance overhead becomes more visible [11]. When performing large sustained I/O performance, user-based file systems achieve comparable performance to kernel file systems [11].

Fuse is a framework for creating user-space file systems for Unix-based operating systems. Since version 2.6.14, Fuse has been a part of the Linux kernel. Fuse can be used to implement a desired file system by utilising an application programming interface (API) composed of file system operations.

Using an application programming interface (API) consisting of file system operations, a desired file system can be programmed with FUSE.

Fuse includes the FUSE kernel module and the libfuse user system library. The reference implementation for communicating with the FUSE kernel module is provided by Libfuse. When an application makes a system call to the FUSE mount point, it is forwarded to the FUSE kernel. The FUSE kernel then forwards the system call to libfuse, which invokes a custom callback for the system call operation. The results of the callback are then returned to libfuse, which passes them through the kernel to the system.

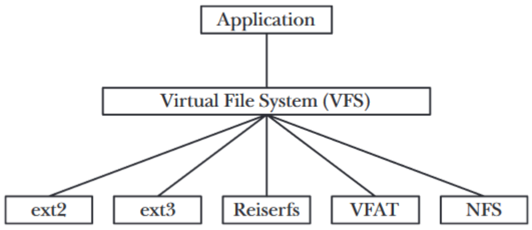

Figure 2.2: The Virtual File System [13]

2.3.2 Malware Interaction with the File System

Applications on the Unix platform are called processes. Processes interact with the file system by making system calls. There are several file system implementations, each with its own specific details. It would be costly for a process to address them all individually. To solve this problem, Linux includes a virtual file system. The virtual file system is an abstraction layer for file system operations, and each file system provides an implementation of the VFS. VFS implements the calls that a file system would implement in a generic interface such as open, read, and write.

fd = open(pathname, flags, mode) opens the file specified by the pathname for reading, writing, or both, depending on the attributes of the flag. The flag also specifies whether the file should be created if it does not already exist. If the open call creates a file, the mode argument specifies the file's permissions. The open system call returns a file descriptor that is used by subsequent system calls to refer to the file.

numread = read(fd, buffer, count) reads bytes from the file referenced by the file descriptor and stores them in the buffer. The read returns the number of bytes read.

numwritten = write(fd, buffer, count) writes the specified number of bytes from the buffer to the file specified by the file descriptor. The write function returns the number of bytes actually written to the file.

2.4 The Electrosense Project

The Electrosense project uses crowdsourcing to collect and analyse spectrum data from low-cost sensors in various parts of the world [14]. Software-defined radio (SDR) front ends and an embedded platform constitute Electorsense sensors. The Raspberry Pi 3 B model is recommended by Electrosense as an embedded platform.

The Raspberry Pi is a credit card-sized, Linux-based, open-source computer board with ARM processors. The Electrosense project uses a Raspbian-based image. Raspbian is a Debian-based operating system designed specifically for the Raspberry Pi hardware.

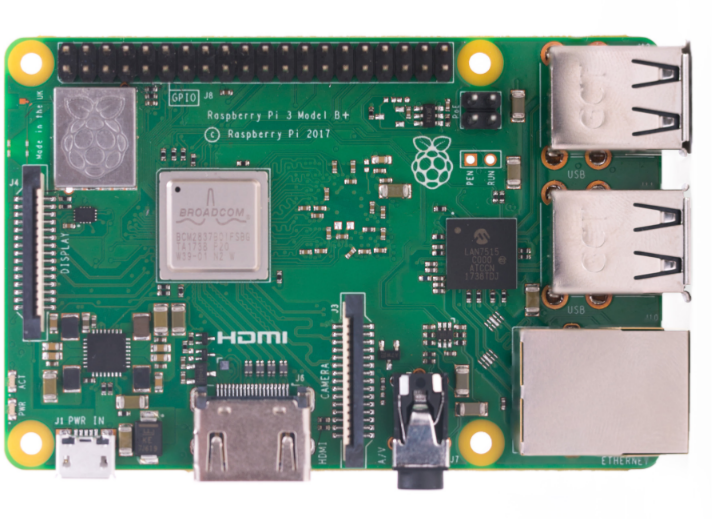

Figure 2.3: Raspberry Pi Model B+ [15]

Figure 2.4: Software-Defined Radio [16]

Chapter 3

Related Work

The purpose of this chapter is to provide a summary of previous works related to our project of using Moving Target Defence techniques to protect against ransomware via the file system. Beginning the chapter is a description of the method used to discover related work. The remaining two sections discuss related work on moving target defence techniques and file systems designed to prevent ransomware.

3.1 Methodology

To ensure the comprehensiveness of the related work, search strategies such as boolean search, the snowballing technique, and domain-specific literature were utilised.

Boolean search is a technique for defining a specific area of interest in combining keywords with the logical operators and, or, and not. The IEEE Explore digital library, the ACM Full-Text Collection, and Springer were used for our literature search. These are the most comprehensive libraries in the field of computer science. "Moving target defence" and "file system" were the Boolean search phrases.

A review of 23 ACM articles/essays (list Appendix A) revealed that none was relevant. None of the twelve articles/essays (list Appendix A) reviewed in IEEE explore was relevant. None of the three Springer articles (list Appendix A) reviewed was relevant. A disadvantage of Boolean search is that the search term may produce false negative results if an article employs the "moving target defence" technique without using the term "moving target defence."

The snowballing technique is a search technique for finding related works from the bibliographies of articles. The articles chosen for this methodology were moving target defence focused surveys [7], [17]–[21]. In reviewing the articles, we determined that none were relevant to our work.

Domain-specific conferences discussing the "moving target defence" technique. The ACM Workshop on MTD was the only conference specifically dedicated to MTD. The conference began in 2014 and is still ongoing. In reviewing 89 papers from conference [22]–[29] no papers relevant to our work were found.

Despite using three distinct techniques to find related work on our specific topic of employing moving target defence techniques on a file system to defend against malware, no such work was found. To gain a better understanding of the subject matter and due to a lack of existing approaches following similar directions, we changed the objective to include related work on moving target defence without the file system component and related work on file system defence against malware without the moving target techniques.

3.2 Related Research on Moving Target Defence

This section discusses how various techniques for moving target defence were implemented.

Dynamic Address Validation Array is a Controller Area Network Bus protocol (CAN Bus). CAN buses can be found in automobiles, robots, and prosthetic devices. CAN enables components to communicate with one another. The CAN bus is unable to implement security mechanisms such as encryption to protect against threats due to the limited hardware capability of the microcontrollers in the CAN bus and the CAN bus eight-byte packet messages protocol. DAVA [30] attempts to mitigate security attacks such as reconnaissance and replay attacks on CAN BUS by employing moving target defence techniques. CAN bus has a manufacturer-determined ID. In a reconnaissance attack, an adversary can reuse an ID to communicate with another component. As a result, an attacker is able to trick another component into performing a task. To prevent such attacks, DAVA periodically modifies the device ID of CAN nodes, preventing the attacker from identifying the devices and preventing communication with them.

Nomad [31] is a system designed to prevent malicious web bot activity in web applications. Malicious web bots engage in a variety of activities, including account registration, spamming comments, and spamming emails. Such activities have detrimental effects on a web application. These activities are carried out by malicious bots, which in the process extract semantic meanings from HTML elements. This semantic includes the name and ID parameter values of HTTP form elements utilised by server-side logic. Due to the dynamic nature of the element, the authors of the paper were able to implement a randomization technique that prevented malicious web bots from obtaining semantic information from such elements. For instance, if the ID of an HTML email field is ”email”, the Noman system will convert this ID to ”random123”. The XRumer, Magic Submitter, SENuke, UCWCS, and Comment Blaster bots have been stopped from performing malicious activities on login pages, thread posting pages, and comment pages by the NOMAD system's randomization method.

The paper [32] presents a method for minimising user tracking through browser fingerprint techniques. The browser fingerprint method derives from the fact that browsers have distinct configurations. Websites may use these distinct configurations to distinguish one browser from another, allowing them to track users for various purposes. The approach of this paper is to modify the browser configuration proactively so that its fingerprint is shared with other browsers. As a result, browser fingerprinting will be less effective, as the configuration of the majority of browsers will be similar.

Solution	*Attack type*	*What to move?*	*How to move?*	*When to move?*	*Implementation ?*
[33], 2011	Unknown attacks	Software Stack (Application Layer, Web Server Layer, OS, Virtualization Layer)	Diversification	Randomly	Ideation
[34], 2011	DoS	Network configuration (ip address and routes)	Shuffling	Fixed period of time	Ideation
[35], 2011	Unknown attacks	Hardware, operating system	Diversification	Triggered by an event, Randomly	Live
[31], 2013	Malicious bots activities (spamming, fake account registration, etc)	HTML elements	Shuffling	Triggered by an event	Live
[36], 2014	Reconnaissance attacks	Operating system	Diversification	Fixed period time	Live
[32], 2019	Browser Fingerprinting	Browser configuration	Shuffling	Triggered by an event	Ideation
[37], 2019	Input based attacks	Algorithm	Diversification	Triggered by an event	Live
[38], 2020	Memory corruption attacks.	Deployment	Redundancy, Diversification	Triggered by an event	Live
[30], 2020	Reconnaissance attack	CAN bus	Shuffling	Fixed period of time	Virtual
[39], 2020	Reconnaissance attacks	AP range	Redundancy	Triggered by an event	Ideation
[39], 2020	Reconnaissance and physical attacks	AP physical location	Redundancy	Triggered by an event	Ideation
[40], 2021	Input based attacks	Model	Diversification	Triggered by an event	Live

Table 3.1: Research Overview Related to Moving Target Defence

The paper [35] presents a design for the Trusted Dynamic Logical Heterogeneity System implementation (TALENT). The objective of Talent is to ensure cyber survivability through platform diversity. It achieves its objectives by sand-boxing the application at the level of the operating system, including the file system, open files, and network connections. The application is then portable checkpoint compiled, making it compatible with multiple architectures and PT platforms. In the paper implementation, 37 combinations were achieved with a migration time of approximately one second.

Multiple Operating System Rotation Environment is proposed and implemented in the paper [36]. MORE is an MTD strategy that rotates multiple operating systems. This technique aims to reduce the likelihood of a reconnaissance attack and restrict the time an attacker has access to a compromised host. It rotates multiple operating systems using virtual machines that host WordPress applications and have different Linux distributions. Using two live and one space IP addresses, the VMS are periodically rotated to become hosts. The results indicate that vulnerabilities can still be exploited in this environment; however, a shorter interval between rotations increases protection against their exploitation. In addition, the effect of the availability of the WordPress application is examined. The test was conducted by using only the metric of packet loss; on average, 2% of packets were lost per minute, and 0.5% per half-minute rotation.

The paper [34] describes the Mutable Networks or Mute. Mute's goal is to prevent DoS attacks from being launched, botnet structures from being created, and network targets from being discovered. It employs random address hopping, in which network systems are assigned new addresses frequently based on random functions. In addition, it utilises a technique known as the ”random fingerprinting approach,” in which responses are altered to provide a false identity for the operating system and applications. MUTE claims to protect critical infrastructure against scanning and DoS attacks.

Moving Attack Surfaces (MAS) is an approach presented in the paper [33] for diversifying web applications. The paper asserts that a single attack surface can be reached and therefore is insecure. The purpose of this paper is to create as many attack surfaces as possible. It proposes creating attack surfaces by diversifying each web stack layer. The stacks would be selected based on the detection of an anomaly or at random. In addition, the stacks will have a finite lifespan, reducing the window of opportunity for an attack. The paper diversifies web application code by implementing the web application layer using various techniques, such as executing a Rails application with two interpreters; ”JRuby” and ”Matz”. It diversifies the Web Server Layer, where it utilises various web server software, such as ”Apache”, ”Nginx”, ”lighttpd”, and ”Tomcat.” It diversifies the operating system layer by using different operating systems such as Windows, Solaris, and Linux. It diversifies the virtualization layer with VMware, Xen, and other hypervisors. The paper concludes that there are 1,356 possible software stacks and attack surfaces for a web server. However, the paper also highlights the difficulties of managing a software stack with such diversity.

The paper [38] describes a moving target defence strategy for dealing with memory corruption attacks. The paper describes a method known as DMON. DMON orchestrates and monitors the execution of a collection of diverse software variants that execute natively on machines with varying instruction sets. A DMON is a particular type of NVX system. An NVX system executes multiple software variants in parallel while monitoring for divergence behaviour. DMON contributes because it executes the different variants on two instruction sets (x86/x64-ARM) and can monitor for divergent behaviour despite the different environments. The research concludes that in the presence of a DMON, attackers must provide a cross-platform exploit to be successful.

The paper [39] proposes a moving target strategy for protecting wireless communications and infrastructure from reconnaissance and targeted attacks (e.g., DOS, eavesdropping). A targeted attack can compromise a specific wireless access point (AP). The author proposes a random range mutation (RNM) technique for AP to prevent such attacks. RNM randomly changes the AP range, compelling wireless clients to switch to a different APN and route, thereby enabling clients to thwart potential attacks. The second protection is based on Random Topology Mutation (RTM). In this instance, a mobile AP is utilised. The authors propose altering the AP's physical location. The authors claim that this facilitates defence against physical attacks and provides similar functionality to RNM, since in this case clients must move to another AP.

Moving target defence techniques are also being explored to protect machine learning models. There are black-box and white-box input-based attacks that reduce the accuracy of certain machine learning models. The authors of the paper [37] propose switching between the algorithms used by the model. The switch is based on a cost model, i.e., if the cost of switching to another algorithm is not beneficial, the switch is not made. The authors base the reasoning on the fact that multiple algorithms can perform a task such as classification, clustering, etc. without a large loss of accuracy, making switching a viable form. The authors test these moving target defence techniques against attacks such as Carlini and Wagner (CW) on a Convolutional Neural Network, Fast Gradient Sign Method against a logistic regression classifier. The authors achieved the worst case accuracy of 0.5 for cases that could bring down the accuracy to 0.

The paper [40] also addresses adversarial attacks on machine learning and uses the ”moving target defence” technique to defend against such attacks. The paper's strategy is to use different models, such that an attack that is successful in one model is less successful in another. They propose to use these models in two ways: randomly or incrementally, as a zero-day attack becomes a frequent attack. They evaluate the approach against attacks such as CW, EAD, and MI-FGSM on the PreAct ResNet18 architecture. The reported baseline accuracy is 94.13%. Papers A and B use different test approaches to evaluate their techniques.

3.3 Related Research on File System Protection Against Ransomware

This paper [41] introduces Rcryptect, a file system designed to detect malicious cryptographic functions at runtime. It attempts to detect malicious behavior by comparing entropy values. In entropy comparison, it is assumed that certain unencrypted file types (MP3, JPG, ZIP) will have a non-random bit stream, as opposed to encrypted files, which will have a random bit stream due to the use of randomness in cryptography. To analyse the entropy as soon as possible, Rcryptect limits the I/O size to 64 KB using the file system. This enables earlier detection of the use of a cryptographic function compared to standard I/O size analysis. After detecting such conduct, the file system removes the ability to write and delete for non-root users. Additionally, it eliminates the suspicious process. The outcome demonstrates that Rcryptect is effective at detecting cryptographic behavior. In the case of RansomEXX, four files were decrypted and two were corrupted prior to detection and termination. JSorm was terminated after encrypting two files. During performance evaluation, the authors observed that writing performance was degraded by less than 10%–15% compared to a naive FUSE file system.

DcyFS [42]is a file system designed to prevent data theft, unauthorised modification, and destruction, to withstand reconnaissance attacks, and to log malicious behavior for later analysis. This is accomplished by implementing a two-layer file system: the base file system and the overlay layer. The overlay layer has many responsibilities, including injecting decoy files, displaying different views of the file system to different processes, and preventing modifications to the base file. Currently, the decoy object is created through a manual process. A policy-driven approach achieves different viewpoints by associating mount namespaces with file system objects and users with security domains. Data integrity is achieved by enforcing a copy-on-write mechanism.

ShieldFS [43] is a layer on top of a native Windows file system. It has two objectives: ransomware detection and file recovery. The detection of ransomware is approached as a supervised classification problem, with a corpus of I/O requests generated by user land code on both infected and uninfected machines. File recovery is possible because files are never lost; instead, ShieldFS writes or deletes the file to a read-only storage area in response to malicious activity. By using the log files, the file system is able to restore the file system to a previous state once the malicious behavior has stopped.

RockFS [44] is a mechanism which offers recovery services capable of undoing unintended operations to files. It achieved this by using a log system. The log system contains all modifications performed to a file. This data is the difference between the new file and the previous one, or if the difference in size is bigger than the file itself. To recover the file, the selective re-execution technique is used, which executes selected operations from the first valid version of the file. The paper assessed that the time to recover grows exponentially with the number of files. For a ransomware attack where there are 10,000 files with 100 versions each, it took the authors 2 hours and 5 minutes to recover every file. Furthermore, the authors concluded that the storage overhead was significant, for instance if users append 10 MB to a file, 10MB will be added to the log.

FITICIO [45] is a stackable file system add-on. Its purpose is to prevent unauthorised file modifications that could corrupt file data. To achieve this, it validates file types whenever a file modification occurs. The file type validation verifies that the file is of the claimed type. It expands on the notion that when malicious behavior such as ransomware encryption occurs, the file is altered to the point where it no longer corresponds to the file type it represents. Fitico's prototype verifies the file's validity by invoking the functionality of popular libraries that support these file types and opening the file with their functionality. As long as the call does not throw errors, the type of the file is considered valid. The authors claim that a disadvantage of the validating file type method is that there is no universal standard for what constitutes a valid file type for some file types, and that some standards, such as txt, can contain any data, making them unvalidatable.

Solution	*Detection*	*Prevention*	*Recovery*
[43], 2016	I/O supervised classification task	None	Read-only replica; Log System
[42], 2018	Decoy objects	Security domains(permissions); Copy-on-Write	None
[44], 2018	None	None	Log System
[45], 2021	File-type validation	None	None
[41], 2022	Entropy analysis	Write, Delete permission	None

Table 3.2: Research Overview Related to File System Protection Against Ransomware

Chapter 4

System Documentation

The system documentation chapter's goal is to document the software development life cycle. The chapter consists of requirement specification and analysis, design of the solution, and implementation of the solution.

4.1 Requirement Specification and Analysis

4.1.1 Ransomware Analysis

To defend against ransomware, we should understand what it does and how it does it. The paper [2] breaks down the malicious interaction between ransomware and the file system into steps: The ransomware scans the file system, encrypts all or a subset of files, and then deletes or overwrites the files.

This paper [46] examines how various ransomware scan the file system. The paper observed that GandCrab and TelsaCrypt traverse the file system directories in depth-first alphabetical order. Similarly, CryptXXX uses depth-first methods, but it traverses directories in reverse alphabetical order. The paper also reveals that depth-first is not the only technique ransomware uses to traverse the file system, as the authors of the paper were unable to identify the traversal pattern of the Osiris and Sage2.2 ransomware. Another traversal mechanism described in [47] in CTB-Locker is ransomware traversing files in size ascending order.

Ransomware encrypts all or a subset of files. The process of encrypting all files can be time-consuming. Paper [47] describes CrpytoLocker and Android Defender as only encrypting file types with specific extensions, such as (pdf, zip). These file formats are commonly referred to as productivity file formats because users utilise them to complete productive tasks. Various techniques, including file extension, specific byte values (magic numbers), and file content analysis, can be used to identify file types [47]. Encoder malware is ransomware that does not encrypt specific file types, but instead encrypts specific directories and file names, such as logs and backups [48].

The paper [47] presents three file interaction patterns for file encryption: Class A, Class B, and Class C. The Class A pattern overwrites a file's contents by opening the file, reading its contents, writing the encrypted content in place, and then closing the file. According to paper [49], a malicious Windows process opens the user's file with the function IRP MJ CREATE, reads the file with the function IRP MJ READ, and then overwrites the file with the function IRP MJ WRITE. Class B moves the user file out of the user's document directory, then reads the content, encrypts the content, and moves the file back to the user's directory. Class C reads the original file, creates a new file with the encrypted data, and deletes the original file. Reading and creation occur independently. The paper [49] demonstrates that the Windows ransom process uses IRP MJ SET INFORMATION to delete the file.

Ransomware provides users with information on how to return files. This document includes fees, due dates, and instructions. The paper [50] demonstrates that the most common ransom note file extension is txt, followed by htm, html, and hta. The papers [47], [49], demonstrate that Teslacrypt creates a ransomware note after exploration of directories. In addition, the paper [50]demonstrates that JSWORM, ChaCha, StopDjvu, LockerGoga, and GlobImposter generate the ransom notes prior to encryption. CryptoLocker and Cerber generate ransom notes after encrypting the directory's contents. In [50], the characteristics of ransom notes, such as the information they contain, the filenames used by the ransom node, and techniques for identifying a ransom note are described in detail.

4.1.2 Requirement Specification

The requirement specification section aims to document the needs of the stakeholders. The stakeholder requirements arise from the concepts of confidentiality, integrity, and availability.

From the analysis, ransomware encrypts files. By encrypting the files, they become inaccessible. Being inaccessible breaches the availability of a system. Availability can be achieved by preventing the encryption of files. **Requirement 1: Prevent the encryption of files.** If prevention can not occur, meaning files are encrypted, then availability can be achieved by recovering files. **Requirement 2: Enable file recovery.**

The ransomware reads the files to encrypt them. By not having the authorization to read the file, the requirement of confidentiality is breached. To protect the requirement of confidentiality, the requirement of preventing ransomware from reading the file arises. **Requirement 3: Prevent ransomware from reading a file.**

Constraints also generate requirements. The objective of this project is to develop a file-system-based solution that utilises the moving target defence technique and runs on a Raspberry Pi 4. **Constraint 1: The solution must be based on a file system implementation. Constraint 2: The solution should employ moving target defence techniques. Constraint 3: The solution should run on a Raspberry Pi 4 device.**

4.2 Design

The design section's objective is to document the transformation of requirements into system design specifications. The chapter includes five design specifications.

4.2.1 Infinite Directory Depth

According to Requirement 1, a solution must prevent file encryption. According to Requirement 3, a solution must prevent ransomware from reading a file. According to an analysis of ransomware behaviour traversal, some ransomware traverses directories in alphabetical order before beginning to read and encrypt files. If the depth of the first alphabetic directory is infinite, ransomware that utilises depth-first traversal will take an infinite amount of time to traverse the directory. If ransomware spends an infinite amount of time traversing the directory, it will not begin encrypting or reading files, as file encryption and reading occur after directory traversal. **Design 1 proposes making the depth of the first alphabetical directory infinite to satisfy Requirements 1 and 3.**

```
|-- !
|   |-- !
|       |-- !
|           |-- !
|               |-- ! (continues)
|-- docs
|   |-- !
|   |   |-- !
|   |       |-- ! (continues)
|   |-- csv.csv
|   |-- doc.doc
|   |-- docx.docx
|-- png.png
|-- pptx.pptx
|-- xls.xls
|-- zip.zip
```

Listing 4.1: Proposed Directory Structure

4.2.2 Changing File Type Identification

According to Requirement 1, a solution must prevent file encryption. According to an analysis of the behavior of ransomware encryption, some ransomware encrypt only certain file types. If a file is of a type in which ransomware has no interest, ransomware will not encrypt it. **Design 2 proposes restricting file type identification to file types that ransomware does not encrypt, thereby meeting Requirement 1.**

4.2.3 Reduced Read and Write Speeds

A solution must prevent file encryption according to Requirement 1. According to an analysis of ransomware's encryption behavior, ransomware reads the file's contents and writes the encrypted version of the read file. Read and write times depend on the performance of the underlying hardware. Encrypting a file will take longer if the read and write speeds are slower than the default read and write speeds. If a file's encryption will take longer than the default, the start of encryption for the following file will be delayed relative to the standard read and write speeds. There is a possibility that ransomware will be detected and stopped by other mechanisms if it persists for an extended period of time. **Design 3 suggests slowing the read and write speeds, thus indirectly meeting Requirement 1.** It indirectly satisfies requirement 1 because some files may not be encrypted if another protection mechanism is in place.

4.2.4 Directory Name Change

According to requirement 1, a solution must prevent file encryption. According to Requirement 2, a solution must prevent ransomware from reading a file. According to an analysis of ransomware encryption behavior, some ransomware targets specific directories, such as administration directories, or files with specific names, such as "backup." If the ransomware discovers a directory with an irrelevant name, it will not encrypt that directory. In addition, ransomware will not encrypt files with a non-relevant name. **Design 4 recommends changing the default file and directory names.** Design 4 satisfies Requirement 1, as neither non-default directory files nor files with non-default names are encrypted. Design 4 meets Requirement 2 because files in non-default directory names and files with no default name are not read.

```
mysql/data -> lorem/ipsum/
mysql/data/mysql -> lorem/ipsum/lorem
share/mysql/charset -> dolor/lorem/charset
share/mysql/english/ -> dolor/lorem/english/
share/mysql/ -> dolor/lorem/
```

Listing 4.2: Changing MySQL Default Directory Names

4.2.5 File Identification Change

According to Requirement 1, a solution must prevent file encryption. According to ransomware encryption pattern C, ransomware reads the original file, creates a new file containing the encrypted data, and then deletes the original file. If the file name is altered after each file is read, the ransomware will be unable to delete the file and will fail to achieve its objective of rendering the file inaccessible. **Design 5 proposes renaming the file after each read, thus satisfying Requirement 1.**

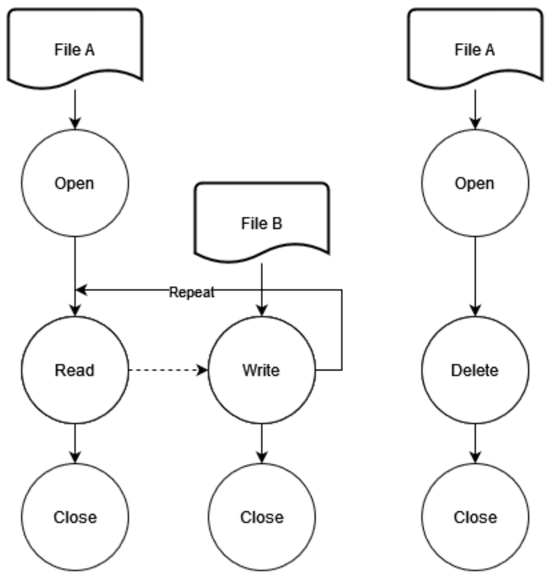

Figure 4.1: Ransomware Encryption Pattern C

4.3 Implementation

The purpose of the implementation section is to document the system's implementation in accordance with the design. Due to time constraints associated with the project, designs 1, 2, and 3 were chosen for implementation. The chapter begins with a discussion of the overlay file system, a fundamental component of all three implementations. The chapter then describes each implementation separately.

4.3.1 Overlay File-System

An overlay file system is a stacked file system with a default file system and an overlay layer. Ext4 is the default file system in the majority of Linux distributions; alternative file systems are available. The overlay layer is a file system that forwards both the request and the response from the base file system to the requesting process. In addition to forwarding the request to the file system, the implementations described in this report modify the request based on design decisions. The same concept also applies to responses. Depending on the design decision, the response may be modified prior to transmission to the process. This is due to the fact that creating a file system from scratch would be a task for which this project lacks the time and resources, and the added value is in implementing the design decisions rather than creating the file system.

The advantage of an overlay file system is that it only implements the essential and value-added features. The majority of the remaining features pertain to the default file system's implementation and performance. This results in a ransomware-aware file system with similar performance to the default file system.

The overlay file system that redirects the request to the underlying file system was developed by the authors of go-fuse/library as a demonstration of how their library functions

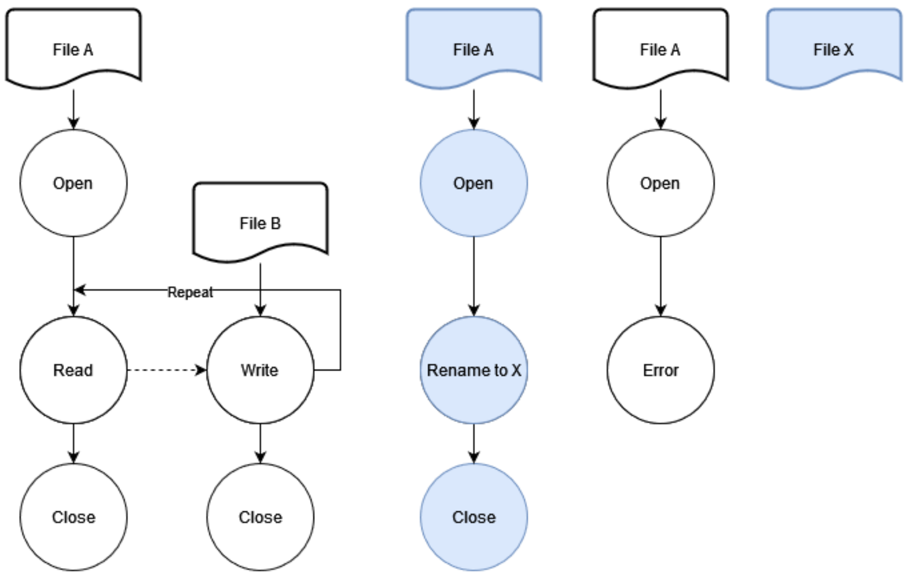

Figure 4.2: Renaming the File After Each Read

and is licensed under the BSD license. Rinor Sefa, the thesis author, is the author of modifications performed to the overlay file system and documented in the report. These modifications implement the design decisions of this project.

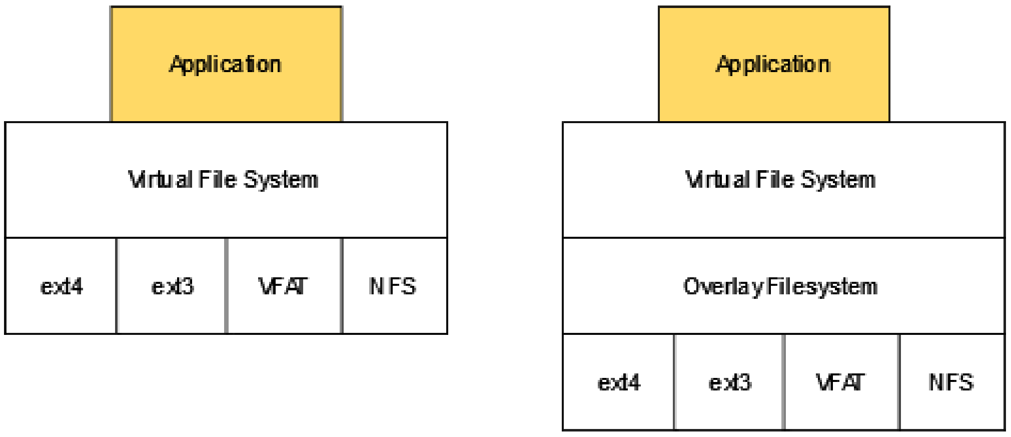

Figure 4.3: Default File System (left) and Overlay File System (right)

4.3.2 Infinite Directory Depth

A process such as ransomware begins its file system traversal at the root directory. To traverse the file system, it must first open the directory in order to retrieve the names of files and directories. To open the directory, the opendir(3) system call is used. Given an input directory name, the opendir() system call returns a pointer to the directory stream. A Dirstream is a structure that stores file and directory names as well as their metadata.

The next step is to invoke the readdir system call, passing the directory stream (DIR) as input. The function returns a pointer to the directory entry. The dir entry contains the next entry, which includes the file i-node number, file or directory name, and, in some file systems, the file type. In situations where the file system does not return the file's type, the lstat(2) function can be invoked.

Design 1 proposes making the depth of the first alphabetical directory infinite. When a process uses the readdir system call to determine the contents of a directory, the overlay layer uses the default file system to create a directory in the directory from which readdir was invoked. The created directory name is set to "!" because "!" is the first alphabetic character in the ASCII sort order format. If the ransomware uses a depth-first search that is sorted alphabetically, it will be trapped in a loop of traversing the directory ! then !/! then !/!/! until it is terminated or runs out of system memory. Thus achieving Design 1's objective.

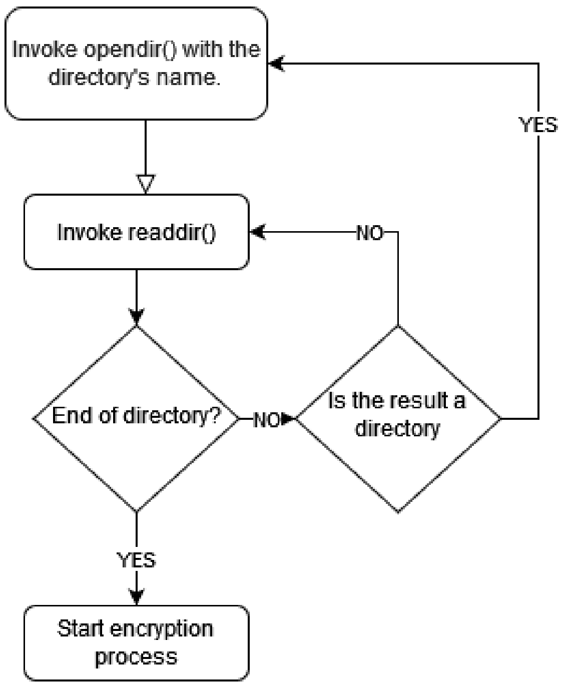
Invoke opendir() with the directory's name.
Invoke readdir()
End of directory?
Is the result a directory
Start encryption process
YES
NO
NO
YES

Figure 4.4: Ransomware BFS traversal

This feature is also extended to ransomware utilising depth-first search with readdir results as the ordering criterion. The filenames returned by readdir() are not in alphabetical order, but rather in the order specified by the underlying file system. To change this, each time the readdir method is invoked, the overlay file system traverses the dir stream, retrieves all the dir entries, and sorts them alphabetically. Then, when the readdir function is invoked, the result will be returned in alphabetical order. Thus, if ransomware uses a depth-first search based on the order of results returned by readdir, it will be trapped in a loop traversing the directory!, then!/!, then!/!/! until it is terminated or runs out of system memory resources. Thus, achieving the objective of Design 1.

The virtual file system must also be modified to achieve the Design 1 specification. The directory entry cache is a feature of the virtual file system. Its purpose is to provide a quick

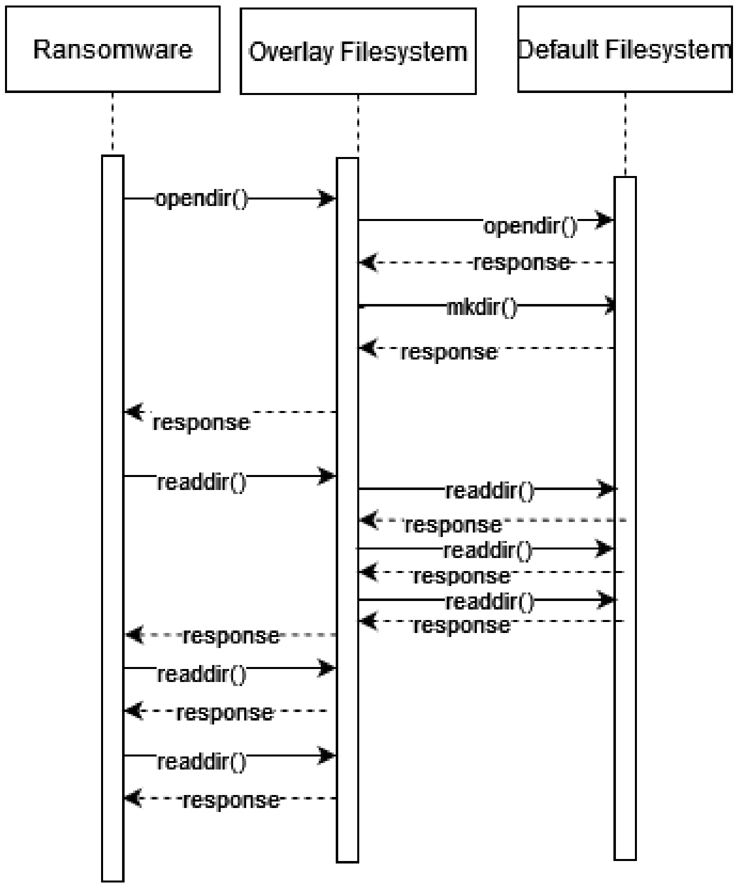

Figure 4.5: Infinite Directory Depth Sequential Diagram

lookup mechanism for translating a path name (file name) to a particular directory entry. Without the directory entry cache, a process wishing to access directory c, which resides in the parent directory a/b/c, would be required to first open directory a, then directory b, and finally directory c. The directory entry cache stores this information in memory to accelerate the process. This Directory Entry Cache will reduce the resources used by the malicious process to discover the file system, allowing encryption to begin. One feature of the directory entry cache is the ability to specify the data's caching duration. This caching time varies based on the file system implementation. In order to address this issue, the overlay file system configures the cache timeout to 1 millisecond. The directory cache is therefore no longer useful.

4.3.3 Reduced Read/Write Speeds

This section describes the implementation of Design 3, which necessitates slowing the read and write speeds.

The read and write speeds of the file are dependent on the underlying hardware. The implementation uses the underlying file system to implement the slowed read and write speeds. A process uses the read() system call to read a file. The process invokes the read system call with the count argument specifying the number of bytes to read.

Typically, the count argument is set to the maximum number of bytes that the underlying hardware can read. After a successful read, the system calls return the number of bytes that were successfully read. The read() function may read fewer bytes than requested. After receiving a read() system call, the overlay file system implementation modifies the count argument from the typical maximum number to a minimum number and forwards the call to the default file system.

The default file system honors the count argument and returns the bytes it reads to the overlay file system. The returned value is passed back to the calling process. As a result, a file that previously required 10 system calls to read can now require 1,000 system calls. This value is dependent on the maximum and minimum bytes that can be read from the underlying device.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);
```

Listing 4.3: Read and Write function synopsis

Similar to the read system call, the write system call also accepts a count argument value specifying how many bytes to read. The overlay file system modifies the count value to the bare minimum and then forwards the request to the underlying default file system. The default file system honors the count argument and returns how many bytes it has read. The number of system calls needed to write 4MB of data has increased from 1 to 4000. Since each system call is an expensive operation, 4000 times more system calls would result in a performance degradation. Design 3 requirements are met as read and write performance degrades.

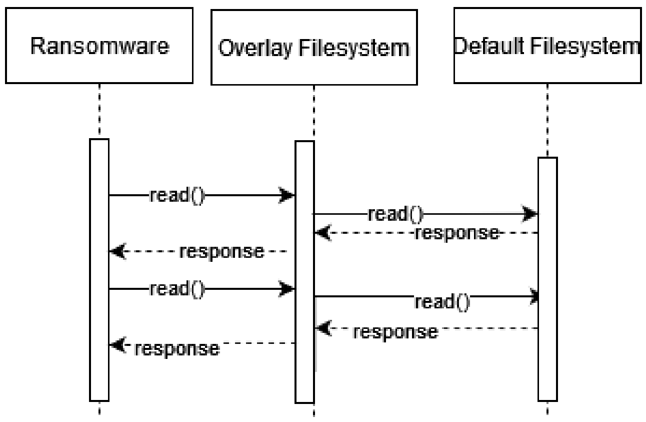

Figure 4.6: Read Flow Chart

The kernel applies optimizations such as clustering disc blocks, allowing processes to access the same file, and sequential read-ahead using the buffer cache. The kernel provides the option to bypass the buffer cache using the O_DIRECT flag when calling the open() system call. Applications, like databases, that have their own caching solutions use this flag. When the O DIRECT flag is set, performance is typically degraded. To implement Design 3, which requires slower read and write speeds, the open() system call is invoked with the O_DIRECT flag.

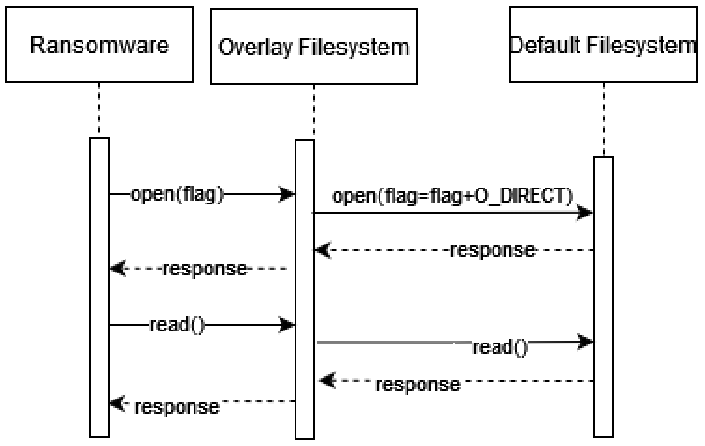

Figure 4.7: Open with O DIRECT flag

Additional adjustments can be made to the virtual file system to decrease the writing speed. The write() system call is asynchronously callable. For example, while one system call is writing the first block of bytes, another system call may be writing the second block of bytes. This results in increased writing speed.

For applications that require synchronous writes, the kernel provides the O-SYNC flag with the open() system call. Calling open with the O-SYNC flag can degrade performance, as the application is blocked until the buffer is written to disc. To implement Design 3, which requires slower write speeds, the open() system call with the O SYNC flag is invoked.

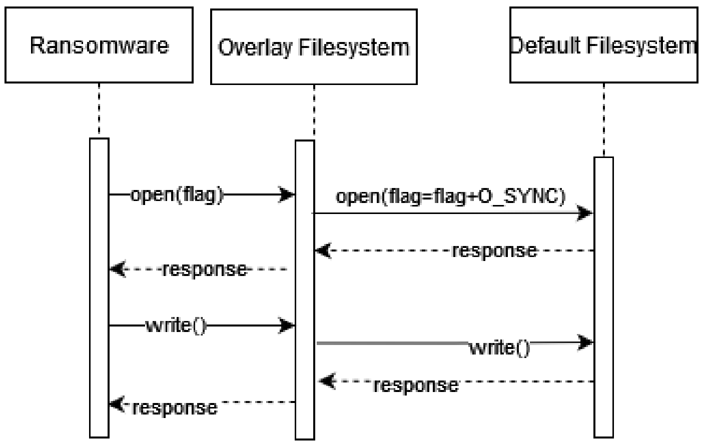

Figure 4.8: Open with O SYNC flag

Modifying the kernel's read-ahead functionality can also result in a decrease in reading speed. The read-ahead functionality ensures that the next block of bytes is read into the buffer before the reading process requires it. On the assumption that a process will read the data sequentially from the lower offset to the higher offset of the files, this enables faster file reading. The size of the read-ahead buffer can be modified within the file system. This value has been set to its minimum to slow down the reading process.

4.3.4 File Type Change

This section describes the implementation of Design 2, which proposes restricting file type identification to file types that ransomware does not encrypt.

As described in the analysis, ransomware typically does not encrypt all files due to the required resources, but instead encrypts ”productive” files to persuade the user into paying the ransom fee. As described in Chapter 42, ransomware and other programs can determine the file type in three ways: file name extension, specific byte values (magic numbers), and analysis of file content.

The file extension or suffix appears at the end of the file name, for example, docx or pdf. A user or an application assigns the file extension, which aids the operating system in determining which application should be used to process the file. Linux does not use file extensions to recognise the file type. File extensions are typically present in the Linux system for cross-platform operability with Windows. Ransomware can determine the file type by using the file extension.

Since ransomware can use the file extension to identify a file type, one solution would be to remove the extension from the file name. The file name's lifecycle is analyzed in order to determine how to remove the extension. When a file is created, it is given its initial name. Open or create system calls can be used to create a file. The create() system call accepts the file's name as a parameter. Before passing the call to the default file system, the overlay file system removes the extension from the file name. Additionally, the file name can be altered after the file has been created. The rename() system call is used to change the file name. The rename system call accepts the new file name as a parameter. Before passing rename() to the default file system, the overlay file system removes the extension from the name. The Design 2 requirements are met by removing the name extension, which can be used to identify the file type.

Magic numbers are bytes within a file that can be used to identify the file type. There are applications such as libmagic, DROID, TrID, and Outside-In that use magic numbers to identify file types. This report examines in greater detail how the open source project Libmagic identifies files in order to determine how to counter it. The file(1) command determines the file type. The file command attempts to recognize signature bytes from the file. The detection uses a database of signature byte-to-file-type mappings. The example file shown below specifies the rules for identifying a PDF file.

```
0           string          %PDF-           PDF document
>5          byte            x               \b, version %c
>7          byte            x               \b.%c
```

Listing 4.4: Example of a PDF file type test

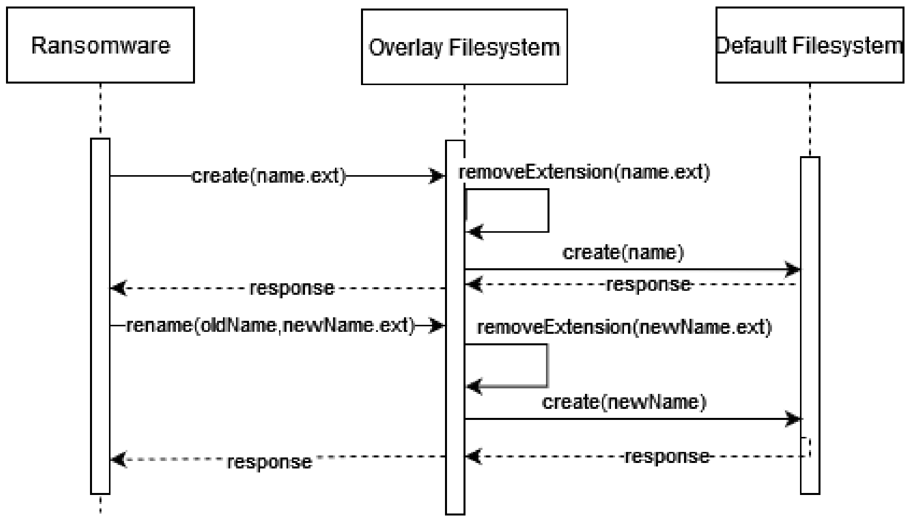

Figure 4.9: Overlay File-System Removing Name Extension

Each line indicates a test to be executed. The line contains a number that specifies the offset, the data type to be tested, the value to be compared, and the message to be printed. The test follows a level approach. The test with no ">" will be evaluated first, followed by ">" and then "> >." Only after the 0 level test has passed can the 1 level test be attempted.

The first line specifies that the application begins reading the file at offset 0 and continues until it encounters a printable character sequence that is at least four characters long and is followed by an unprintable character. The application then verifies whether the bytes match the string %PDF-. If this condition is met, the application prints the file as a PDF document. The following test will occur at the offset: 5. It examines whether this one-byte value is equal to x. In this case, X indicates any value. The application then prints the PDF document version.

```
00000000   25 50 44 46  2D 31 2E 33  %PDF−1.3
00000008   0D 0A 25 E2  E3 CF D3 0D  ..%.....
00000010   0A 0D 0A 31  20 30 20 6F  ...1 0 o
00000018   62 6A 0D 0A  3C 3C 0D 0A  bj..<<..
00000020   2F 54 79 70  65 20 2F 43  /Type /C
00000028   61 74 61 6C  6F 67 0D 0A  atalog..
00000030   2F 4F 75 74  6C 69 6E 65  /Outline
00000038   73 20 32 20  30 20 52 0D  s 2 0 R.
```

Listing 4.5: Hex viewer of a PDF file

For demonstration purposes, the identification of a sample PDF file's content is explained. The first row of listing 4.5 represents the binary representation of the offset, the second row represents the byte representation in hex, and the third row represents the hex-to-text conversions. The first test will determine whether the first four printable characters followed by a non-printable character are PDF-. Since this is true, "PDF document" will be printed. The second and third tests will check offsets 5 and 7 to determine the PDF file's version number and print 1.0.

Assuming that the malware uses a similar method to recognise file types an approach to defend against it would be to not provide these magic numbers. Malware and other processes read these numbers by utilising the read() system call to read the file's contents. The second parameter of the read system call is the buffer, which is filled with bytes from the file after the read() command is executed. In the preceding example, a portion of the buffer would contain information about the magic numbers that could be used to identify the file type.

Based on this information, the malware would decide whether or not to encrypt the file. Before filling the buffer with data, it must be determined if it contains information of this file type. If it contains file type information, it should not be included in the buffer. As a result, the malware will be unaware of the file's extension. The alternative strategy would consist of filling this buffer with file type information for a file type that malware would not be interested in encrypting, such as Linux system file types (ELF, BIN).

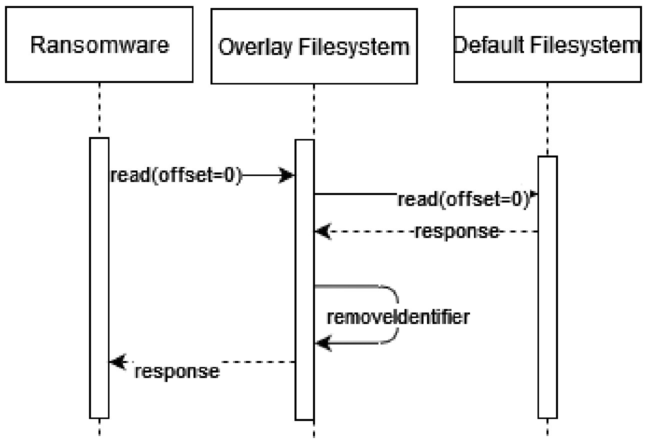

Figure 4.10: Overlay File-system Removing Identifier Flow Chart

Content analysis is an additional method for identifying file types. Unlike magic numbers, which typically appear at the beginning of a file or at specific offsets, content analysis examines a portion of the file or the entire file. This analysis is more resource-intensive and time-consuming than identification by magic numbers. For example, txt and CVS files do not contain magic numbers.

To determine if a file is txt, an application like libmagic verifies that the file contains readable characters, such as those defined in the ASCII standard. Or, it verifies whether CSV files adhere to the format specified in RFC-4180 and RFC-7111. The solution strategy suggested for the magic numbers could also be applied to this problem with the adjustment that, while for magic numbers only reads beginning at offset 0 were analysed, for content analysis all reads should be analysed.

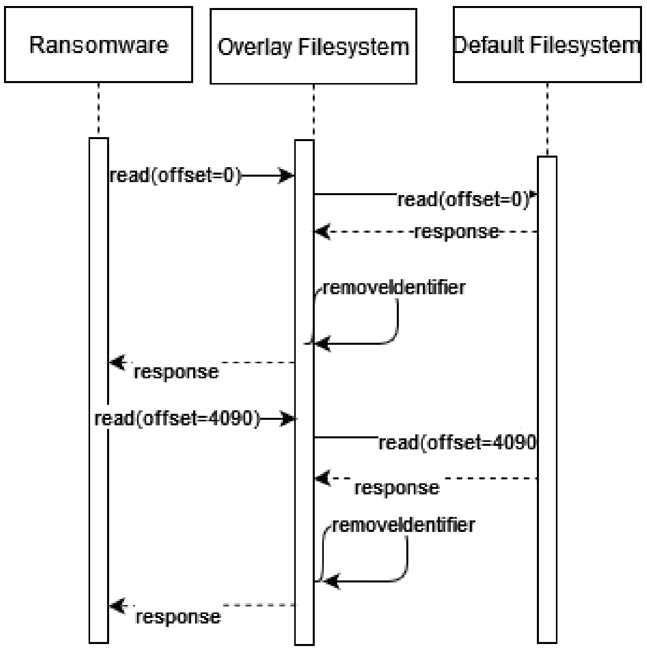

Figure 4.11: Overlay File-System Removing Identifier on Each Read Flow Chart

Chapter 5

Evaluation

The evaluation chapter discusses whether the implemented file systems conform to the requirement's specification. Additionally, it describes how the performance of the implemented file systems compares to that of the default file systems.

5.1 Hardware and Software Specification

The evaluation is conducted using a Raspberry Pi 4 B Rev 1.4. The Raspberry Pi board includes a 64 GB class 10 microSD card with a performance rating of 10. At the time of evaluation, 11G of the 63G storage capacity was occupied. The system software is Debian GNU/Linux 11.

5.1.1 File Corpus

Files are encrypted by ransomware. A set of files is required to evaluate the implementations presented in the report. The chosen data set is the Govdocs1 data set. The Govdocs1 corpus is a publicly available data set containing one million documents. One million documents may not be manageable, so a subset of data is selected for evaluation. The selected subset of data consists of 1000 randomly selected files from the Govdocs1 data set. The advantage of using Govdocs1 is that it facilitates evaluation repetition because the data is publicly available. It permits evaluation comparison with other Govdocs1 implementations. It closely resembles a potential attack environment due to the techniques used to collect the data set.

5.1.2 Ransomware Selection

Historically, malware primarily affected Microsoft Windows-based computers. Various types of devices are currently under attack by malware. These devices have different CPU architectures, different hardware, different operating systems, and different libraries. Consequently, malware or software designed for one device may not function on another device due to a lack of available resources on that platform. Two ransomware known to target Raspberry Pi 4 devices are examined during the evaluation. DarkRadiation is a Bash-written ransomware that targets Linux operating systems. It targets files with.txt,.sh,.py, and database extensions and encrypts all /home directory files. The RansomPoc is a Python-based proof-of-concept ransomware that targets files with certain extension.

5.2 Result

In this section, the results of an evaluation based on two criteria are presented: lost files and encrypting time.

RansomwarePoc successfully encrypted 946 out of 979 files on the default file system and took 7 minutes to complete. DarkRadiation successfully encrypted all files on the default file system and took 1 minute 37 seconds to complete.

RansomwarePoc, against the infinite directory file system implementation. RanspomPoc was unable to encrypt files because it terminated itself while traversing the directories. When RanspomPoc reached a depth of 955 directories, it terminated with a recursion error. A recursion error is a Python error that is thrown when a function is called more than the default recursion limit of 1000 times. The number 1000 is close to the temporary directory depth of 995 at which the recursion error occurred. The total time from application start to application termination was 10 minutes. The creation of 995 directories resulted in the storage of 3.9 MiB of metadata.

DarkRadiation was similarly incapable of encrypting files in the infinite file system. DarkRadiation was trapped while traversing the endless directories. Following a 31-minute traversal of DarkRadiation, 1005 directories with a memory size of 3.9 MB were produced.

RansomPoc, against the ”slow write and read speed” implementation. RansomPoc was able to encrypt 1 byte of all files. Based on RansomPoc's dynamic analysis, files are encrypted using the Class A encryption pattern. The Class A pattern overwrites the contents of a file by opening the file, reading the contents, writing the encrypted contents to the location, and then closing it. RansomPoc performed only one read and one write per file. The ”slow read and write speed” implementation reduces the read and write size to 1 byte; consequently, RansomPoC encrypted only the first byte of the file. Since only the first byte of each file was encrypted, file recovery is possible. The RansomPoc took four seconds to execute.

DarkRadiation was able to encrypt all files on the slow read-write file system. DarkRadiation followed the class B encryption pattern. The Class B encryption pattern moves the file from the original directory, then reads the contents, encrypts the contents, and moves the encrypted file back to the user's directory. DarkRadiation moved the file out of the read-write file system and then returned the encrypted version of the file to the original

Ransomware	*Implementation*	*Lost files*	*Time*
RansomPoc	Default file system	946/979	7m 51s
	Infinite-directory file system	0	10m 32s
	File-type file system	0	4s
	Slow-read-speed	0	4s
DarkRadiation	Default file system	976/979	1m 14s
	Infinite-directory file system	0	31m
	File-type file system	0	1m 50s
	Slow-read-write file system	976/979	15m 7s

Table 5.1: Evaluating File System Protection Against Ransomware

directory. We cannot verify that the returned file is a result of encrypting the original file. All encrypted files had a size of 32 bytes, which differs from the original file sizes.

RansomPoc was unable to encrypt any files against the file-type file system. Based on dynamic analysis, RansomPoc encrypts files with specific file name extensions. RansomPoc was not interested in encrypting files because the implementation of the file-type file system prohibits file names with extensions. The execution of the RansomPoc took four seconds.

DarkRadiation was able to encrypt all files located in the file type file system. DarkRadiation's static analysis revealed that DarkRadiation encrypts all files under the home directory. Since the file type file system is mounted under the /home directory, DarkRadiation is able to encrypt all files. If the mount point of the file type is changed outside the home directory, DarkRadiation is not able to encrypt files. DarkRadiation cannot encrypt files because DarkRadiation is only interested in files with certain extensions outside the /home directory. Since the file type file system removes extensions from file names, DarkRadiation loses interest in encrypting files if they are outside the home directory.

5.3 Performance Overhead

To measure the performance overhead of the developed file systems, the IOzone benchmark is utilised. The IOzone is a specialised tool that analyses the performance of a file system using various load generation and file access patterns.

The MicroSD disc standard specification guarantees I/O speeds of 10 MB/s and a maximum boost speed of 25 MB/s. With a ”small” file size of 16384 kilobytes, the read speeds for the default file systems can reach up to 2450 MB/s, while the infinite directory file system and the file type file system can reach up to 1164 MB/s and 817 MB/s, respectively. We believe performance mechanisms such as buffer cache, processor cache, read-ahead, etc. are responsible for these significant differences in performance compared to the MicroSD specification. Once the file size approaches the RAM size, as in the case of a 4 GB file, the reading speed begins to approach the hardware specification. The reading speed of the default file system is 22 MB/s, while the infinite directory and file type file systems reach 21 MB/s. In terms of read performance for small file sizes, the default file system outperforms the infinite directory file system and the file type file system by a factor of 2.1 and 2.9, respectively. The CPU utilisation data for read performance reveals that the default file system utilises the CPU at 96.9%, whereas the infinite directory file system and the file type file system only utilise the CPU at 61.87% and 65.03%, respectively. The read speeds of the infinite directory file system and the file-type file system are comparable, with the infinite directory file system exhibiting better performance. We believe the difference is due to the additional analysis performed during the first read operation on the file-type file system. During the first read system call, the file type checks to see if the first block has identification bytes.

The evaluation data for write performance indicates that all file systems exceed the microSD specification performance of 25 MB/s for small files. The default file system achieves a speed of 326 MB/s for a file size of 16394 kilobytes, whereas the infinite directory file system and file type file system achieve speeds of 155 MB/S and 157 MB/S, respectively. We believe that this is because the write system call is called asynchronously. Once the file size reaches 4GB, the maximum performance of the default file system is 15 MB/s, while the maximum performance of the infinite and file type file systems is 15 MB/s and 14 MB/s, respectively. When the file size is small, the write performance of the default file system is 2.1 times greater than that of the infinite directory file system and the file type file system. The performance of the write system call is similar for files with a size of 4 GB.

The ZIP file includes the other IOZone measurements (re-read, re-write, read backwards, random-read, random-write, backward read, record rewrite, stride read, fwrite, re-write, and fe-fread) using different file sizes and record sizes. As the objective of the project was not to develop a file system focused on performance, the remaining results are not discussed.

Note: Note: The IOzone was unable to test the performance of slow-read-write file systems because the record size could not be altered due to design decision.

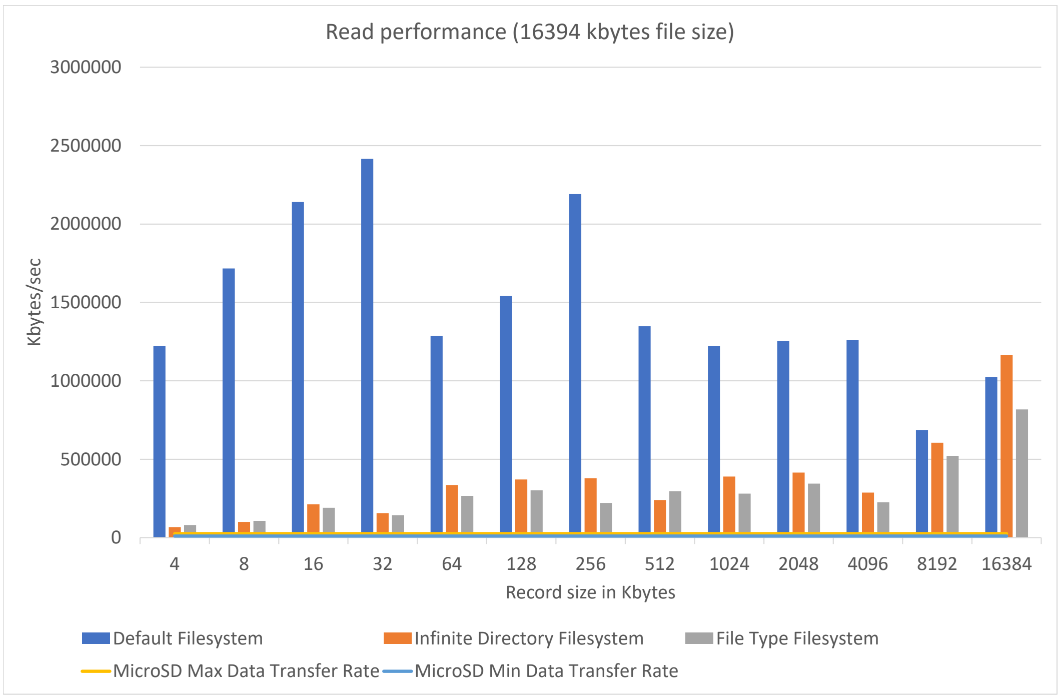

Figure 5.1: Read Performance (16MB File Size)

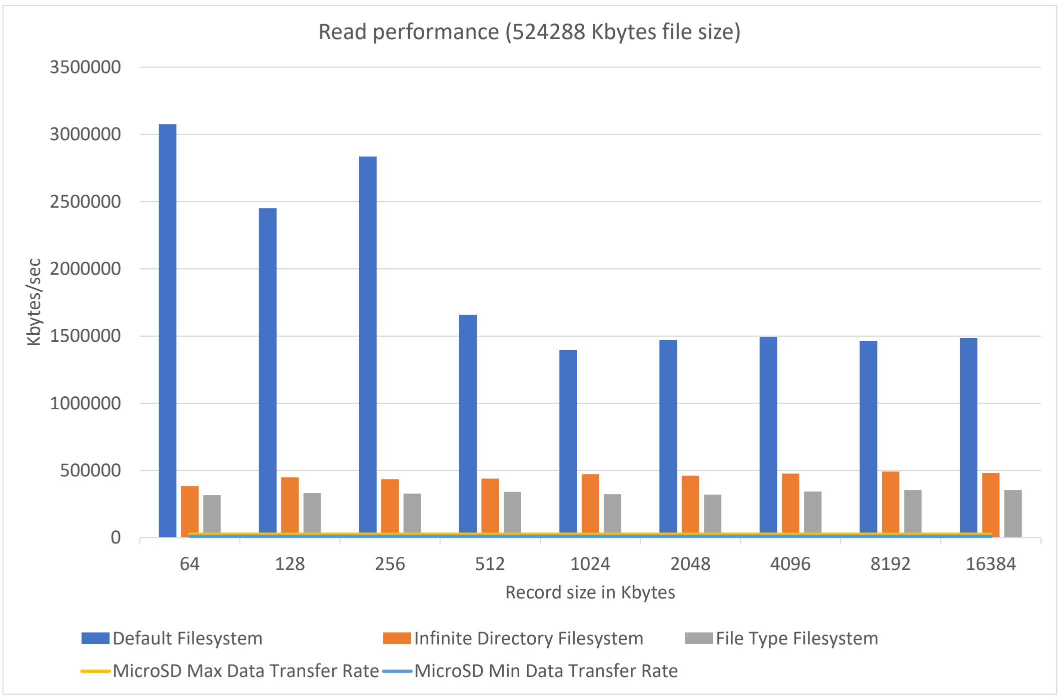

Figure 5.2: Read Performance (524MB File Size)

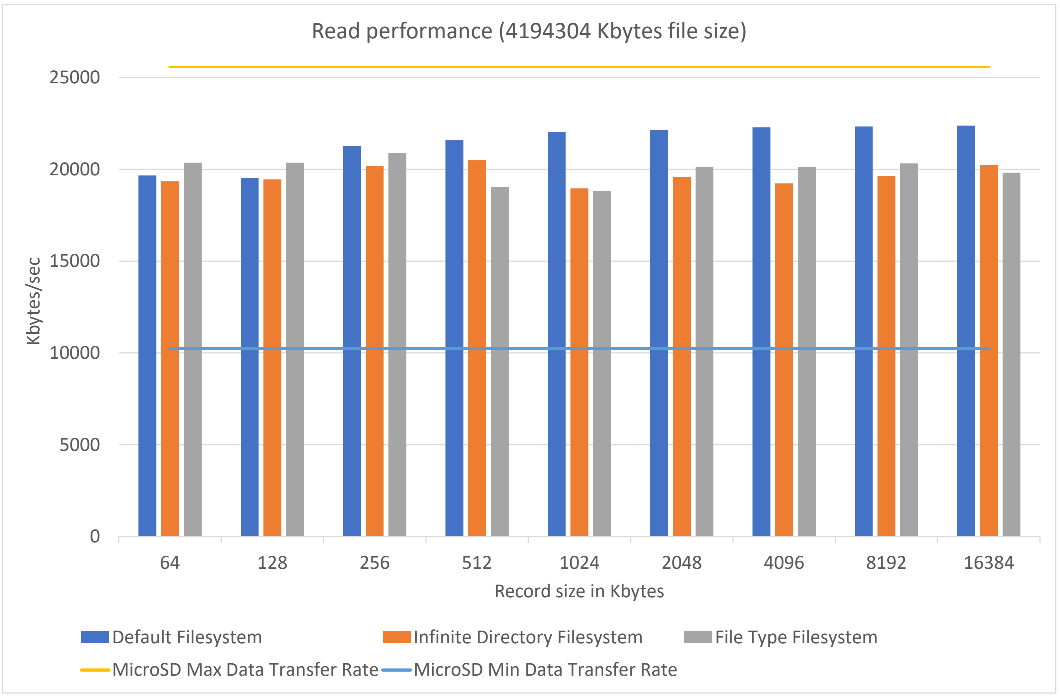

Figure 5.3: Read Performance (4GB File Size)

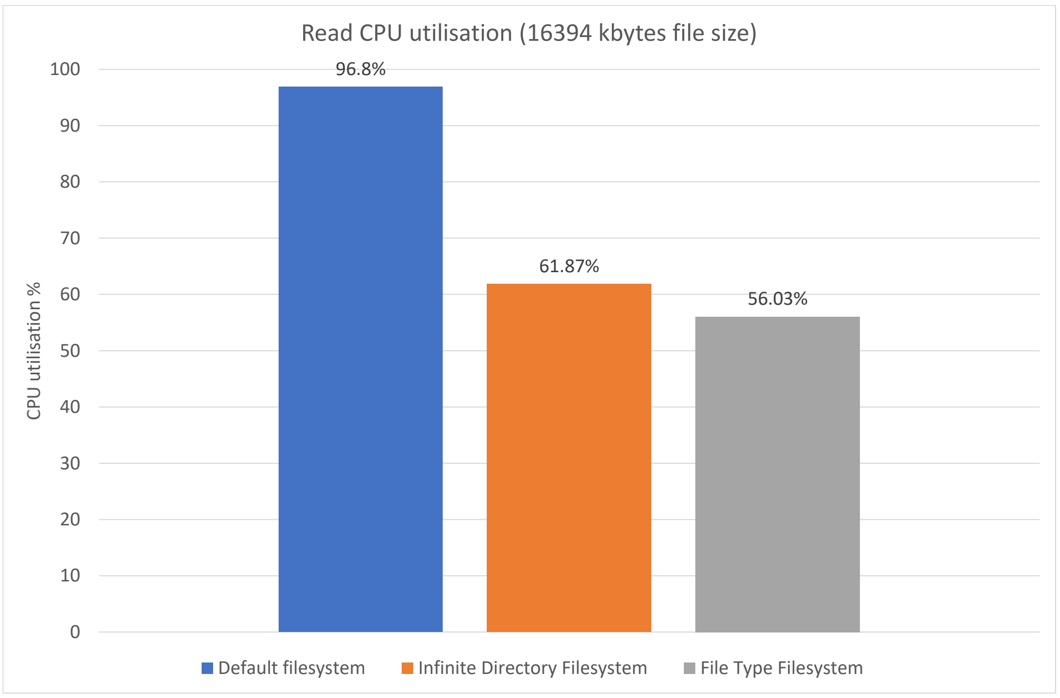

Figure 5.4: Read CPU Utilisation (16MB File Size)

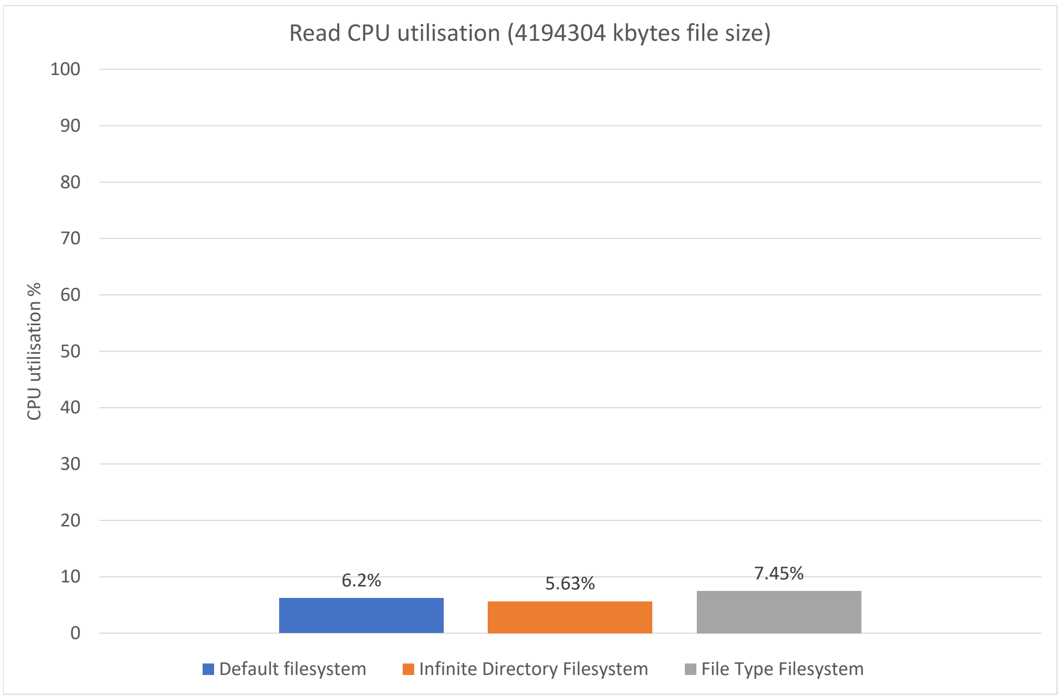

Figure 5.5: Read CPU Utilisation (4GB File Size)

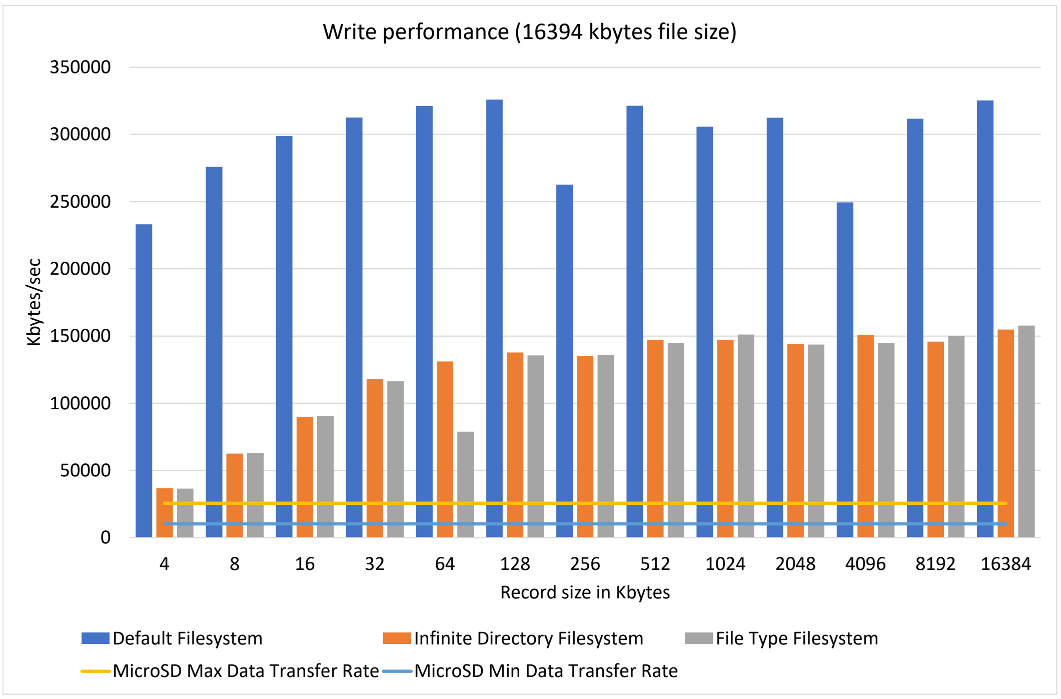

Figure 5.6: Write Performance (16MB File Size)

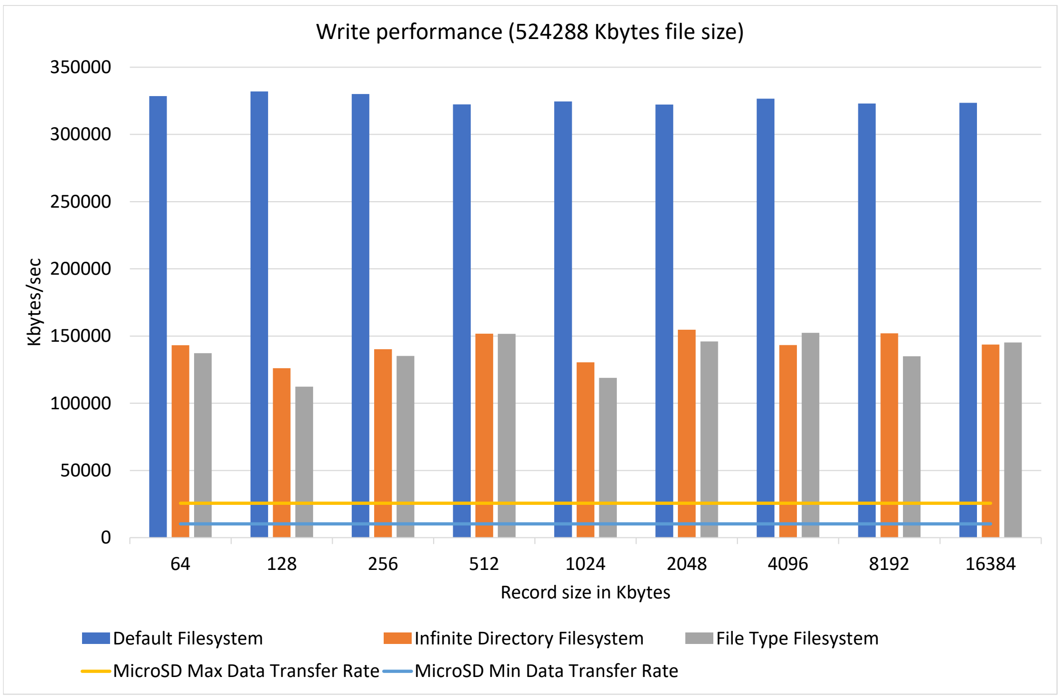

Figure 5.7: Write Performance (524MB File Size)

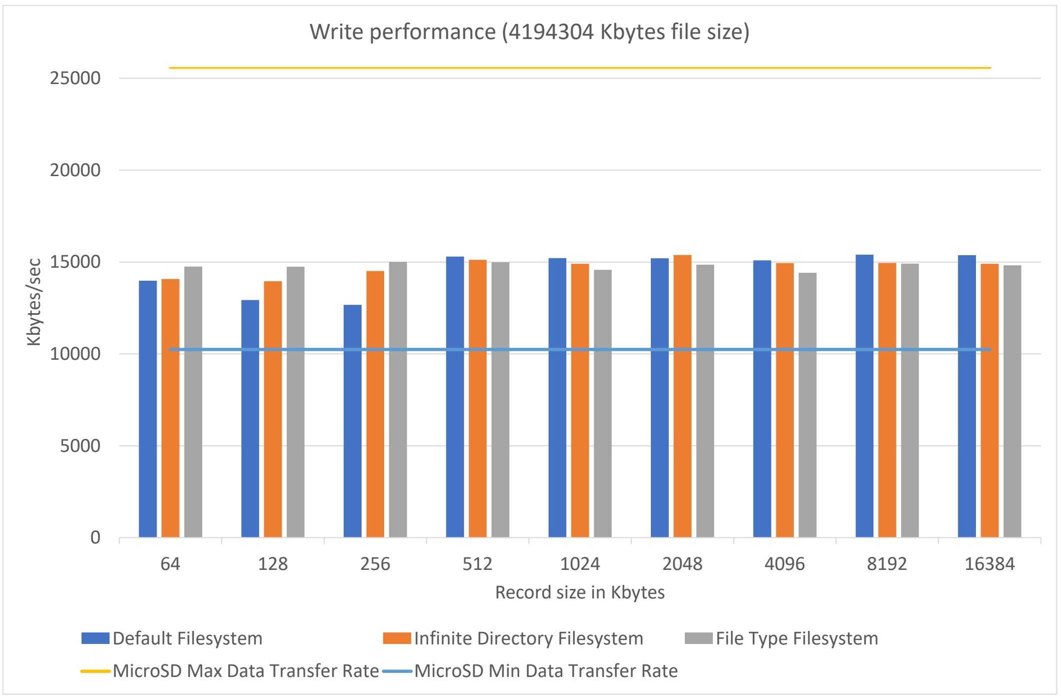

Figure 5.8: Write Performance (4GB File Size)

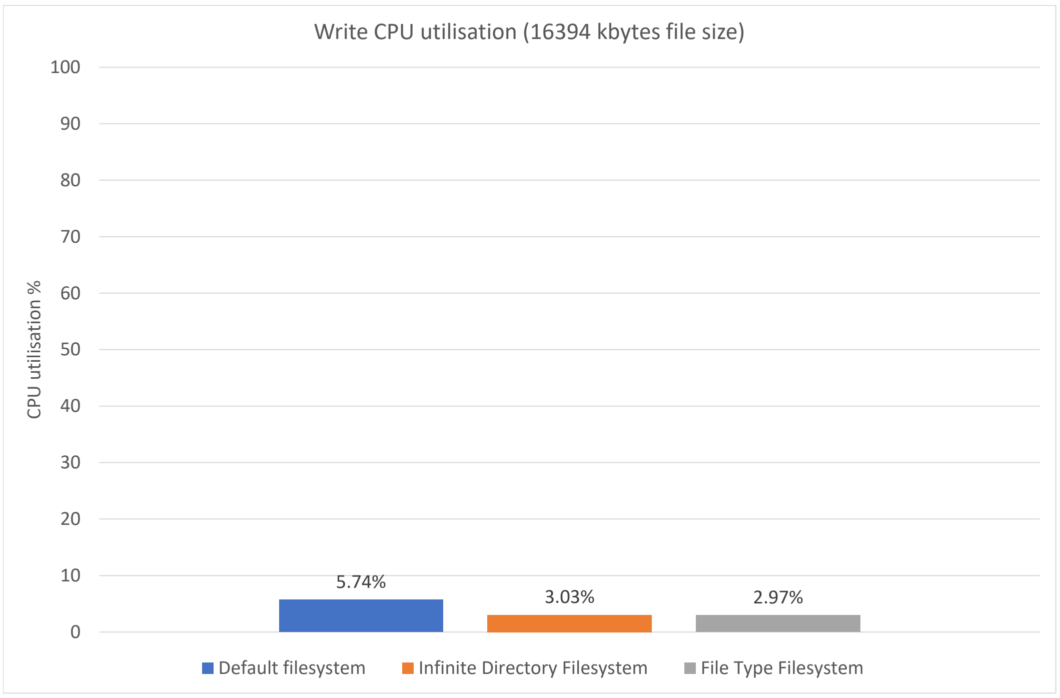

Figure 5.9: Write CPU Utilisation (16MB File Size)

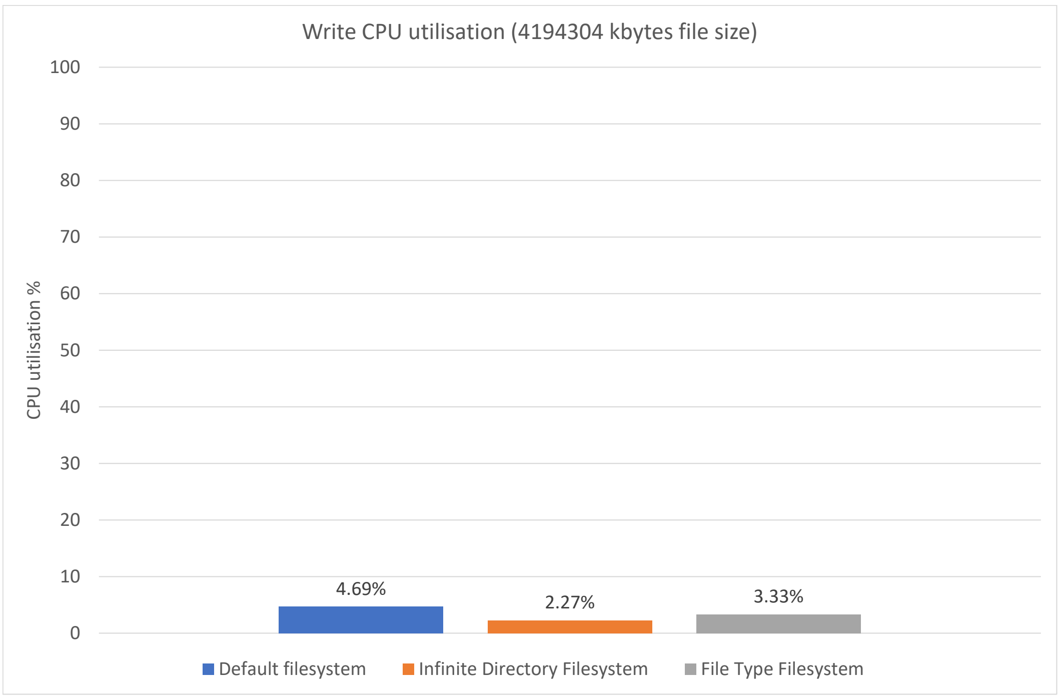

Figure 5.10: Write CPU Utilisation (4GB File Size)

Chapter 6

Limitations, Future Research

6.1 Limitations

The performance of file systems is not evaluated on an SD card that is empty and has a fresh operating system installation. The performance of the current file system is evaluated on an SD card containing additional data. Evaluations that are not conducted in a clean environment make it difficult for others to reproduce. In addition, the SD card's performance may suffer if it is not empty.

The evaluation compares different file systems. In the file type file system, for example, there are two implementations: one corresponds to removing the file's extension and the other corresponds to removing the file's identity bits. It is difficult to determine which feature, the extension remover or the bit remover, prevented file encryption by comparing file systems. For a more comprehensive evaluation, we would recommend comparing the file system implementations.

The evaluation revealed that the developed file systems are inferior to the default file system in terms of performance when the file size was small. This is due to the file system's underlying structure. If file systems are developed from the ground up with performance requirements, performance of the implemented file systems can be enhanced.

The implemented file systems cannot differentiate between malicious and benign processes. Thus, a benign application may become trapped in the infinite directory trap or be unable to open a file because it does not recognise the file's extension. To address this issue, it may be necessary to modify the applications that interact with the file system, or to add an access control mechanism to the file system.

The file systems implemented were only evaluated against two ransomware. More ransomware should be considered during the evaluation phase for a more reliable assessment.

6.2 Recommendations for Future Research

During the period of research, only a few articles analysing ransomware on Linux were identified. The majority of ransomware analysis research focuses on ransomware that runs on the Microsoft Windows operating system. Additionally, since Linux platforms are also ransomware targets, additional research is needed to comprehend how ransomware interacts with Linux platforms.

Due to time and resource constraints, only 3 out of 5 design specifications were implemented. Both the directory name change and file identifier change design specifications would protect files from ransomware like DarkRadiation. If the directory name change were to be implemented, DarkRadiation would not be able to encrypt all files under the home directory because there would be no home directory. Since DarkRadiation implements encryption pattern C, the design specification for the file identifier change could have protected the original files from being deleted by DarkRadiation. We suggest considering whether it is worth implementing these designs in future projects.

When evaluating ransomware, we had to rely on manual review of log files to understand how RansomPoc and DarkRadiation perform traversal, encryption, and selection tasks. We believe researchers can save a lot of time by automating the log review task.

Chapter 7

Conclusion

The increasing use of IoT devices in a variety of projects that contain valuable data makes IoT devices a target for ransomware. Due to their limited resources, IoT devices cannot implement resource-intensive ransomware protection mechanisms. This work implements a light-way protection mechanism comprised of overlay file systems that use moving target defence techniques to defend against ransomware.

The file type overlay file system removes file-type-indicating elements. These elements are present in the file name extension and file signature bytes. When evaluated against RansomPoc and DarkRadiation ransomware, the file type file system successfully prevented all files from being encrypted.

The Infinite Directory file system uses an infinite directory to trap ransomware that use a depth-first traversal strategy. Both the RansomPoc and DarkRadiation ransomware were trapped while traversing the infinite directory and failed to encrypt any files.

The slow-read-write file system reduces the encryption speed by changing the parameters of read and write system calls and minimizes the performance enhancement of the file system. The goal of the Slow-Read-Write file system is to increase the encryption time of files. DarkRadiation encrypted all files in 15 minutes and 6 seconds, compared to 1 minute and 14 seconds in the default file system. In the case of RansomPoc, the encrypted files could be recovered because the ransomware encrypted only the first byte of the files, compared to 4096 bytes in the default file system.

The main limitation of implemented file systems is that they cannot distinguish between malicious and non-malicious applications. Consequently, non-malicious applications can also be impacted by the file system's features. In addition, the implemented file systems are not fast compared to the default file system for reading and writing small file sizes. The performance of the default file system is up to 2.9 times faster than the implemented file systems when reading small files and up to 2.1 times faster when writing small files.

Bibliography

[1] M. Bishop, E. Sullivan, and M. Ruppel, *Computer security: art and science*, en, Second edition. Boston: Addison-Wesley, 2019, OCLC: on1076675266, ISBN: 978-0-321-71233-2.

[2] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, „A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions“, *ACM Computing Surveys*, Jan. 2022, Just Accepted, ISSN: 0360-0300. DOI: `10.1145/3514229`. [Online]. Available: `http://doi.org/10.1145/3514229` (visited on 05/16/2022).

[3] *Treasury Continues to Counter Ransomware as Part of Whole-of-Government Effort; Sanctions Ransomware Operators and Virtual Currency Exchange*, en. [Online]. Available: `https://home.treasury.gov/news/press-releases/jy0471` (visited on 09/25/2022).

[4] *Treasury Takes Robust Actions to Counter Ransomware*, en. [Online]. Available: `https://home.treasury.gov/news/press-releases/jy0364` (visited on 09/25/2022).

[5] „National Cyber Leap Year Summit 2009“, en, p. 58,

[6] T. Yadav and R. A. Mallari, „Technical Aspects of Cyber Kill Chain“, en, *arXiv:1606.03184 [cs]*, vol. 536, pp. 438–452, 2015, arXiv: 1606.03184. DOI: `10.1007/978-3-319-22915-7_40`. [Online]. Available: `http://arxiv.org/abs/1606.03184` (visited on 05/08/2022).

[7] B. C. Ward, S. R. Gomez, R. Skowyra, *et al.*, „Survey of Cyber Moving Targets Second Edition“, en, MIT Lincoln Laboratory Lexington United States, Tech. Rep., Jan. 2018. [Online]. Available: `https://apps.dtic.mil/sti/citations/AD1055276` (visited on 04/26/2022).

[8] R. Zhuang, S. A. DeLoach, and X. Ou, „Towards a Theory of Moving Target Defense“, en, in *Proceedings of the First ACM Workshop on Moving Target Defense - MTD '14*, Scottsdale, Arizona, USA: ACM Press, 2014, pp. 31–40, ISBN: 978-1-4503-3150-0. DOI: `10.1145/2663474.2663479`. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2663474.2663479` (visited on 04/20/2022).

[9] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, „Dynamic Malware Analysis in the Modern Era—A State of the Art Survey“, en, *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–48, Sep. 2020, ISSN: 0360-0300, 1557-7341. DOI: `10.1145/3329786`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3329786` (visited on 04/27/2022).

[10] E. Gandotra, D. Bansal, and S. Sofat, „Malware Analysis and Classification: A Survey“, en, p. 9,

[11] A. Rajgarhia and A. Gehani, „Performance and extension of user space file systems“, en, in *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, Sierre, Switzerland: ACM Press, 2010, p. 206, ISBN: 978-1-60558-639-7. DOI: `10.1145/1774088.1774130`. [Online]. Available: `http://portal.acm.org/citation.cfm?doid=1774088.1774130` (visited on 05/09/2022).

[12] B. K. R. Vangoor, V. Tarasov, and E. Zadok, „To FUSE or Not to FUSE: Performance of User-Space File Systems“, en, p. 15,

[13] M. Kerrisk, *The Linux programming interface: a Linux and UNIX system programming handbook*, en. San Francisco: No Starch Press, 2010, ISBN: 978-1-59327-220-3.

[14] S. Rajendran, R. Calvo-Palomino, M. Fuchs, *et al.*, „Electrosense: Open and Big Spectrum Data“, en, *IEEE Communications Magazine*, vol. 56, no. 1, pp. 210–217, Jan. 2018, ISSN: 0163-6804. DOI: `10.1109/MCOM.2017.1700200`. [Online]. Available: `http://ieeexplore.ieee.org/document/8121869/` (visited on 05/10/2022).

[15] *Teach, learn, and make with the Raspberry Pi Foundation*, en. [Online]. Available: `https://www.raspberrypi.org/` (visited on 10/03/2022).

[16] *RTL-SDR Blog R820T2 RTL2832U 1PPM TCXO SMA Software Defined Radio with Dipole Antenna*, en-US. [Online]. Available: `https://www.rtl-sdr.com/product/rtl-sdr-blog-r820t2-rtl2832u-1ppm-tcxo-sma-software-defined-radio-with-dipole-antenna-kit/` (visited on 10/03/2022).

[17] R. E. Navas, F. Cuppens, N. Boulahia Cuppens, L. Toutain, and G. Z. Papadopoulos, „MTD, Where Art Thou? A Systematic Review of Moving Target Defense Techniques for IoT“, en, *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7818–7832, May 2021, ISSN: 2327-4662, 2372-2541. DOI: `10.1109/JIOT.2020.3040358`. [Online]. Available: `https://ieeexplore.ieee.org/document/9270287/` (visited on 04/20/2022).

[18] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, „Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense“, en, *arXiv:1909.08092 [cs]*, Sep. 2019, arXiv: 1909.08092. [Online]. Available: `http://arxiv.org/abs/1909.08092` (visited on 04/20/2022).

[19] G.-l. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, „Moving target defense: State of the art and characteristics“, en, *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, Nov. 2016, ISSN: 2095-9184, 2095-9230. DOI: `10.1631/FITEE.1601321`. [Online]. Available: `http://link.springer.com/10.1631/FITEE.1601321` (visited on 04/20/2022).

[20] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, „Moving Target Defense Techniques: A Survey“, en, *Security and Communication Networks*, vol. 2018, pp. 1–25, Jul. 2018, ISSN: 1939-0114, 1939-0122. DOI: `10.1155/2018/3759626`. [Online]. Available: `https://www.hindawi.com/journals/scn/2018/3759626/` (visited on 04/20/2022).

[21] J. Zheng and A. S. Namin, „A Survey on the Moving Target Defense Strategies: An Architectural Perspective“, en, *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, Jan. 2019, ISSN: 1000-9000, 1860-4749. DOI: `10.1007/s11390-019-1906-z`. [Online]. Available: `http://link.springer.com/10.1007/s11390-019-1906-z` (visited on 04/20/2022).

[22] *MTD '14: Proceedings of the First ACM Workshop on Moving Target Defense*, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, ISBN: 9781450331500.

[23] *MTD '15: Proceedings of the Second ACM Workshop on Moving Target Defense*, Denver, Colorado, USA: Association for Computing Machinery, 2015, ISBN: 9781450338233.

[24] *MTD '16: Proceedings of the 2016 ACM Workshop on Moving Target Defense*, Vienna, Austria: Association for Computing Machinery, 2016, ISBN: 9781450345705.

[25] *MTD '17: Proceedings of the 2017 Workshop on Moving Target Defense*, Dallas, Texas, USA: Association for Computing Machinery, 2017, ISBN: 9781450351768.

[26] *MTD '18: Proceedings of the 5th ACM Workshop on Moving Target Defense*, Toronto, Canada: Association for Computing Machinery, 2018, ISBN: 9781450360036.

[27] *MTD'19: Proceedings of the 6th ACM Workshop on Moving Target Defense*, London, United Kingdom: Association for Computing Machinery, 2019, ISBN: 9781450368285.

[28] *MTD'20: Proceedings of the 7th ACM Workshop on Moving Target Defense*, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380850.

[29] *MTD '21: Proceedings of the 8th ACM Workshop on Moving Target Defense*, Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, ISBN: 9781450386586.

[30] R. Brown, A. Marti, C. Jenkins, and S. Shannigrahi, „Dynamic Address Validation Array (DAVA): A Moving Target Defense Protocol for CAN bus“, in *Proceedings of the 7th ACM Workshop on Moving Target Defense*, ser. MTD'20, New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 11–19, ISBN: 978-1-4503-8085-0. DOI: `10.1145/3411496.3421221`. [Online]. Available: `http://doi.org/10.1145/3411496.3421221` (visited on 05/24/2022).

[31] S. Vikram, C. Yang, and G. Gu, „NOMAD: Towards non-intrusive moving-target defense against web bots“, in *2013 IEEE Conference on Communications and Network Security (CNS)*, Oct. 2013, pp. 55–63. DOI: `10.1109/CNS.2013.6682692`.

[32] A. Gómez-Boix, D. Frey, Y.-D. Bromberg, and B. Baudry, „A Collaborative Strategy for Mitigating Tracking through Browser Fingerprinting“, in *Proceedings of the 6th ACM Workshop on Moving Target Defense*, ser. MTD'19, New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 67–78, ISBN: 978-1-4503-6828-5. DOI: `10.1145/3338468.3356828`. [Online]. Available: `http://doi.org/10.1145/3338468.3356828` (visited on 05/24/2022).

[33] Y. Huang and A. K. Ghosh, „Introducing Diversity and Uncertainty to Create Moving Attack Surfaces for Web Services“, en, in *Moving Target Defense*, S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., vol. 54, Series Title: Advances in Information Security, New York, NY: Springer New York, 2011, pp. 131–151. DOI: `10.1007/978-1-4614-0977-9_8`. [Online]. Available: `http://link.springer.com/10.1007/978-1-4614-0977-9_8` (visited on 06/20/2022).

[34] E. Al-Shaer, „Toward Network Configuration Randomization for Moving Target Defense“, en, in *Moving Target Defense*, S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., vol. 54, Series Title: Advances in Information Security, New York, NY: Springer New York, 2011, pp. 153–159. DOI: 10.1007/978-1-4614-0977-9_9. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-0977-9_9 (visited on 06/20/2022).

[35] H. Okhravi, A. Comella, E. Robinson, S. Yannalfo, P. Michaleas, and J. Haines, „Creating a Cyber Moving Target for Critical Infrastructure Applications“, en, in *Critical Infrastructure Protection V*, J. Butts and S. Shenoi, Eds., vol. 367, Series Title: IFIP Advances in Information and Communication Technology, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 107–123. DOI: 10.1007/978-3-642-24864-1_8. [Online]. Available: http://link.springer.com/10.1007/978-3-642-24864-1_8 (visited on 06/20/2022).

[36] M. Thompson, N. Evans, and V. Kisekka, „Multiple OS rotational environment an implemented Moving Target Defense“, in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*, Aug. 2014, pp. 1–6. DOI: 10.1109/ISRCS.2014.6900086.

[37] A. Roy, A. Chhabra, C. A. Kamhoua, and P. Mohapatra, „A moving target defense against adversarial machine learning“, en, in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, Arlington Virginia: ACM, Nov. 2019, pp. 383–388, ISBN: 978-1-4503-6733-2. DOI: 10.1145/3318216.3363338. [Online]. Available: https://dl.acm.org/doi/10.1145/3318216.3363338 (visited on 08/02/2022).

[38] A. Voulimeneas, D. Song, F. Parzefall, *et al.*, „Distributed Heterogeneous N-Variant Execution“, en, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Maurice, L. Bilge, G. Stringhini, and N. Neves, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 217–237, ISBN: 978-3-030-52683-2. DOI: 10.1007/978-3-030-52683-2_11.

[39] Q. Duan, E. Al-Shaer, and J. Xie, „Range and Topology Mutation Based Wireless Agility“, in *Proceedings of the 7th ACM Workshop on Moving Target Defense*, ser. MTD'20, New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 59–67, ISBN: 978-1-4503-8085-0. DOI: 10.1145/3411496.3421228. [Online]. Available: http://doi.org/10.1145/3411496.3421228 (visited on 05/24/2022).

[40] S. Abdelnabi and M. Fritz, „What's in the box: Deflecting Adversarial Attacks by Randomly Deploying Adversarially-Disjoint Models“, in *Proceedings of the 8th ACM Workshop on Moving Target Defense*, New York, NY, USA: Association for Computing Machinery, Nov. 2021, ISBN: 978-1-4503-8658-6. [Online]. Available: http://doi.org/10.1145/3474370.3485659 (visited on 05/24/2022).

[41] S. Lee, N.-s. Jho, D. Chung, Y. Kang, and M. Kim, „Rcryptect: Real-time detection of cryptographic function in the user-space filesystem“, en, *Computers & Security*, vol. 112, p. 102 512, Jan. 2022, ISSN: 01674048. DOI: 10.1016/j.cose.2021.102512. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404821003369 (visited on 08/16/2022).

[42] T. Taylor, F. Araujo, A. Kohlbrenner, and M. P. Stoecklin, „Hidden in Plain Sight: Filesystem View Separation for Data Integrity and Deception“, en, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Giuffrida, S. Bardin, and G. Blanc, Eds., vol. 10885, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, pp. 256–278. DOI: `10.1007/978-3-319-93411-2_12`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-93411-2_12` (visited on 07/30/2022).

[43] A. Continella, A. Guagnelli, G. Zingaro, *et al.*, „ShieldFS: The Last Word In Ransomware Resilient Filesystems“, en, p. 20,

[44] D. R. Matos, M. L. Pardal, G. Carle, and M. Correia, „RockFS: Cloud-backed File System Resilience to Client-Side Attacks“, en, in *Proceedings of the 19th International Middleware Conference*, Rennes France: ACM, Nov. 2018, pp. 107–119, ISBN: 978-1-4503-5702-9. DOI: `10.1145/3274808.3274817`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3274808.3274817` (visited on 08/02/2022).

[45] T. McIntosh, P. Watters, A. Kayes, A. Ng, and Y.-P. P. Chen, „Enforcing situation-aware access control to build malware-resilient file systems“, en, *Future Generation Computer Systems*, vol. 115, pp. 568–582, Feb. 2021, ISSN: 0167739X. DOI: `10.1016/j.future.2020.09.035`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/S0167739X20305641` (visited on 08/02/2022).

[46] L. Grant and S. Parkinson, „Identifying File Interaction Patterns in Ransomware Behaviour“, en, in *Guide to Vulnerability Analysis for Computer Networks and Systems*, S. Parkinson, A. Crampton, and R. Hill, Eds., Series Title: Computer Communications and Networks, Cham: Springer International Publishing, 2018, pp. 317–335. DOI: `10.1007/978-3-319-92624-7_14`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-92624-7_14` (visited on 07/29/2022).

[47] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, „CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data“, en, in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Nara, Japan: IEEE, Jun. 2016, pp. 303–312, ISBN: 978-1-5090-1483-5. DOI: `10.1109/ICDCS.2016.46`. [Online]. Available: `http://ieeexplore.ieee.org/document/7536529/` (visited on 08/01/2022).

[48] *Linux.Encoder. - Wikipedia*, en. [Online]. Available: `https://en.wikipedia.org/wiki/Linux.Encoder.` (visited on 10/04/2022).

[49] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, „Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks“, en, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Almgren, V. Gulisano, and F. Maggi, Eds., vol. 9148, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2015, pp. 3–24. DOI: `10.1007/978-3-319-20550-2_1`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-20550-2_1` (visited on 07/30/2022).

[50] Y. Lemmou, J.-L. Lanet, and E. M. Souidi, „In-Depth Analysis of Ransom Note Files“, en, *Computers*, vol. 10, no. 11, p. 145, Nov. 2021, ISSN: 2073-431X. DOI: `10.3390/computers10110145`. [Online]. Available: `https://www.mdpi.com/2073-431X/10/11/145` (visited on 08/16/2022).

List of Figures

List of Tables

Appendix A

Appendix A

A.1 ACM Full- Text Collection

Z. Gu, B. Saltaformaggio, X. Zhang, and D. Xu, Guest-transparent honey files via hypervisor-level access redirection, Comput. Secur., vol. 77, no. C, pp. 737–744, Aug. 2018, doi: 10.1016/j.cose.2018.02.014.

P. Larsen and M. Franz, "Adoption Challenges of Code Randomization," in Proceedings of the 7th ACM Workshop on Moving Target Defense, New York, NY, USA, Nov. 2020, pp. 45–49. doi: 10.1145/3411496.3421226.

Z. Shu and G. Yan, "Ensuring Deception Consistency for FTP Services Hardened against Advanced Persistent Threats," in Proceedings of the 5th ACM Workshop on Moving Target Defense, New York, NY, USA, Jan. 2018, pp. 69–79. doi: 10.1145/3268966.3268971.

S. Lee, H. K. Kim, and K. Kim, "Ransomware protection using the moving target defense perspective," Comput. Electr. Eng., vol. 78, no. C, pp. 288–299, Sep. 2019, doi: 10.1016/j.compeleceng.2019.07.014.

J. Sun, S. Liu, and K. Sun, "A Scalable High Fidelity Decoy Framework against Sophisticated Cyber Attacks," in Proceedings of the 6th ACM Workshop on Moving Target Defense, New York, NY, USA, Nov. 2019, pp. 37–46. doi: 10.1145/3338468.3356826.

C. E. Rubio-Medrano, J. Lamp, A. Doupé, Z. Zhao, and G.-J. Ahn, "Mutated Policies: Towards Proactive Attribute-based Defenses for Access Control," in Proceedings of the 2017 Workshop on Moving Target Defense, New York, NY, USA, Oct. 2017, pp. 39–49. doi: 10.1145/3140549.3140553.

H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions," ACM Comput. Surv., Jan. 2022, doi: 10.1145/3514229.

C. A. Odell, M. R. McNiece, S. K. Gage, H. D. Gage, and E. W. Fulp, "Using Probability Densities to Evolve more Secure Software Configurations," in Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense, New York, NY, USA, Oct. 2015, pp. 27–32. doi: 10.1145/2809826.2809831.

N. Nissim, O. Lahav, A. Cohen, Y. Elovici, and L. Rokach, “Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud,” Comput. Secur., vol. 87, no. C, Nov. 2019, doi: 10.1016/j.cose.2019.101590.

R. Moussaileb, N. Cuppens, J.-L. Lanet, and H. L. Bouder, “A Survey on Windows-based Ransomware Taxonomy and Detection Mechanisms,” ACM Comput. Surv., vol. 54, no. 6, p. 117:1-117:36, Jul. 2021, doi: 10.1145/3453153.

S. Liu, P. Feng, S. Wang, K. Sun, and J. Cao, “Enhancing malware analysis sandboxes with emulated user behavior,” Comput. Secur., vol. 115, no. C, Apr. 2022, doi: 10.1016/j.cose.2022.102613.

H. Kurra, Y. Al-Nashif, and S. Hariri, “Resilient cloud data storage services,” in Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, New York, NY, USA, Aug. 2013, pp. 1–9. doi: 10.1145/2494621.2494634.

M. S. Kashkoush, M. Azab, G. Attiya, and A. S. Abed, “Online Smart Disguise: real-time diversification evading coresidency-based cloud attacks,” Cluster Computing, vol. 22, no. 3, pp. 721–736, Sep. 2019, doi: 10.1007/s10586-018-2851-2.

R. Jolak et al., “CONSERVE: A framework for the selection of techniques for monitoring containers security,” J. Syst. Softw., vol. 186, no. C, Apr. 2022, doi: 10.1016/j.jss.2021.111158.

D. J. John, R. W. Smith, W. H. Turkett, D. A. Cañas, and E. W. Fulp, “Evolutionary based moving target cyber defense,” in Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, Jul. 2014, pp. 1261–1268. doi: 10.1145/2598394.2605437.

O. Hireche, C. Benzaïd, and T. Taleb, “Deep data plane programming and AI for zero-trust self-driven networking in beyond 5G,” Comput. Netw., vol. 203, no. C, Feb. 2022, doi: 10.1016/j.comnet.2021.108668.

X. Han, N. Kheir, and D. Balzarotti, “Deception Techniques in Computer Security: A Research Perspective,” ACM Comput. Surv., vol. 51, no. 4, p. 80:1-80:36, Jul. 2018, doi: 10.1145/3214305. [14] G. Gu, H. Hu, E. Keller, Z. Lin, and D. E. Porter, “Building a Security OS With Software Defined Infrastructure,” in Proceedings of the 8th Asia-Pacific Workshop on Systems, New York, NY, USA, Sep. 2017, pp. 1–8. doi: 10.1145/3124680.3124720.

K. Falzon and E. Bodden, Dynamically Provisioning Isolation in Hierarchical Architectures, in Proceedings of the 18th International Conference on Information Security - Volume 9290, Berlin, Heidelberg, Sep. 2015, pp. 83–101. doi: 10.1007/978-3-319-23318-55.

B. Coppens, B. De Sutter, and S. Volckaert, “Multi-variant execution environments,” in The Continuing Arms Race: Code-Reuse Attacks and Defenses, vol. 18, Association for Computing Machinery and Morgan AND Claypool, 2018, pp. 211–258. Accessed: May 16, 2022.

M. Barbareschi, A. De Benedictis, and N. Mazzocca, “A PUF-based hardware mutual authentication protocol,” J. Parallel Distrib. Comput., vol. 119, no. C, pp. 107–120, Sep. 2018, doi: 10.1016/j.jpdc.2018.04.007.

A. Bajic and G. T. Becker, “Automated benchmark network diversification for realistic attack simulation with application to moving target defense,” Int. J. Inf. Secur., vol. 21, no. 2, pp. 253–278, Apr. 2022, doi: 10.1007/s10207-021-00552-9.

C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, “From Security to Assurance in the Cloud: A Survey,” ACM Comput. Surv., vol. 48, no. 1, p. 2:1-2:50, Jul. 2015, doi: 10.1145/2767005.

A.2 IEE

T. Kong, L. Wang, D. Ma, K. Chen, Z. Xu, and Y. Lu, “ConfigRand: A Moving Target Defense Framework against the Shared Kernel Information Leakages for Container-based Cloud,” in 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Dec. 2020, pp. 794–801. doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00104.

J. Sianipar, M. Sukmana, and C. Meinel, “Moving Sensitive Data Against Live Memory Dumping, Spectre and Meltdown Attacks,” in 2018 26th International Conference on Systems Engineering (ICSEng), Dec. 2018, pp. 1–8. doi: 10.1109/ICSENG.2018.8638178.

Y. Zhang, D. Ma, X. Sun, K. Chen, and F. Liu, “What You See Is Not What You Get: Towards Deception-Based Data Moving Target Defense,” in 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), Nov. 2020, pp. 1–8. doi: 10.1109/IPCCC50635.2020.9391522.

T. Masumoto, W. K. Kyi Oo, and H. Koide, “MTD: Run-time System Call Mapping Randomization,” in 2021 International Symposium on Computer Science and Intelligent Controls (ISCSIC), Nov. 2021, pp. 257–263. doi: 10.1109/ISCSIC54682.2021.00054.

R. Biswas and J. Wu, “Protecting Resources Against Volumetric and Non-volumetric Network Attacks,” in 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), Dec. 2021, pp. 387–395. doi: 10.1109/ICPADS53394.2021.00054.

S. Huang and J. Pan, “A Software MTD Technique of Multipath Execution Protection,” in 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), Sep. 2020, pp. 489–496. doi: 10.1109/ICISCAE51034.2020.9236800.

W. K. Kyi Oo, H. Koide, D. Vasconcellos Vargas, and K. Sakurai, “A New Design for Evaluating Moving Target Defense System,” in 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), Nov. 2018, pp. 561–563. doi: 10.1109/CANDARW.2018.00111.

G.-C. Luh and W.-W. Liu, “Potential Field Based Immune Network for Dynamic Motion Planning of Mobile Robots,” in 2006 International Forum on Strategic Technology, Oct. 2006, pp. 151–155. doi: 10.1109/IFOST.2006.312275.

F. Mohsen and H. Jafaarian, “Raising the Bar Really High: An MTD Approach to Protect Data in Embedded Browsers,” in 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Jul. 2019, vol. 1, pp. 786–794. doi: 10.1109/COMPSAC.2019.00116.

X. Tao, F. Esposito, A. Sacco, and G. Marchetto, “A Policy-Based Architecture for Container Migration in Software Defined Infrastructures,” in 2019 IEEE Conference on Network Softwarization (NetSoft), Jun. 2019, pp. 198–202. doi: 10.1109/NETSOFT.2019.8806659.

M. Thompson, M. Mendolla, M. Muggler, and M. Ike, “Dynamic Application Rotation Environment for Moving Target Defense,” in 2016 Resilience Week (RWS), Aug. 2016, pp. 17–26. doi: 10.1109/RWEEK.2016.7573301.

B. Tozer, T. Mazzuchi, and S. Sarkani, “Optimizing Attack Surface and Configuration Diversity Using Multi-objective Reinforcement Learning,” in 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Dec. 2015, pp. 144–149. doi: 10.1109/ICMLA.2015.144.

A.3 Springer

G. Li, W. Wang, K. Gai, Y. Tang, B. Yang, and X. Si, “A Framework for Mimic Defense System in Cyberspace,” J. Signal Process. Syst., vol. 93, no. 2–3, pp. 169–185, Mar. 2021, doi: 10.1007/s11265-019-01473-6.

M. S. Kashkoush, M. Azab, G. Attiya, and A. S. Abed, “Online Smart Disguise: real-time diversification evading coresidency-based cloud attacks,” Cluster Computing, vol. 22, no. 3, pp. 721–736, Sep. 2019, doi: 10.1007/s10586-018-2851-2.

A. Bajic and G. T. Becker, “Automated benchmark network diversification for realistic attack simulation with application to moving target defense,” Int. J. Inf. Secur., vol. 21, no. 2, pp. 253–278, Apr. 2022, doi: 10.1007/s10207-021-00552-9.