University of Zurich[UZH]

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

Detection and Classification of Malware using File System Dimensions for MTD on IoT

Róbert Oles
Zurich, Switzerland
Student ID: 17-736-653

Supervisor: Dr. Alberto Huertas Celdrán, Jan von der Assen, Chao Feng
Date of Submission: May 3, 2023

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/

Zusammenfassung

Das Ziel dieser Masterarbeit ist der Entwurf und die Implementierung eines Systems, das dynamisch Ransomware basierend auf der Dateisystemaktivitäten erkennt. Die Implementierung eines Overlay-Dateisystems hat es ermöglicht, die Dateisystemaktivität aller Prozesse in Form von einer kommaseparierten Datei (CSV) zu erfassen. Merkmale wie die Entropie von Schreibvorgängen, Anzahl von Lesevorgängen und die Anzahl von Schreiboperationen werden verwendet, um die Klassifizierungsmodelle zu trainieren. Weiter dient das Overlay-Dateisystem auch dem Zweck, den Angriff abzuschwächen. Sobald das Modell böswillige Aktivitäten erkennt, initiiert das Overlay-Dateisystem eine Moving Target Defense Strategie (MTD), die den Namen einer Datei ändert, nachdem die Datei von einem Prozess gelesen wurde. Dadurch werden Änderungen an der Datei verhindert, was die Ransomware davon abhält, weitere Benutzerdaten zu verschlüsseln. Zusätzlich werden die gesammelten Daten in Bezug auf die Entropie von Schreibvorgängen sowie die Anzahl der Lese- und Schreibvorgänge von unterschiedliche Arbeitslasten analysiert. Die Dateisystemoperationen für bösartige und gutartige Workloads werden interpretiert. Weiter wurde das Erkennungssystem auf einem Raspberry Pi bereitgestellt. Die vorgeschlagene Lösung demonstrierte eine hohe Leistung in Bezug auf Erkennungsgeschwindigkeit und Genauigkeit. Schliesslich wurde der Leistungsaufwand des Erkennungssystems analysiert. Bei laufendem Erkennungssystem hat sich die Geschwindigkeit der Schreiboperationen verdoppelt, im Vergleich zur laufenden Maschine ohne Erkennungssystem.

Abstract

The aim of this thesis is the design and implementation of a system that dynamically detects Ransomware based on file system activity. Implementation of custom overlay file system has made possible to log the file system activity of all processes in a form of comma-separated values (CSV) file. Features such as entropy of write operations, number of reads and number of write operations are used to train the classification models. Further, the overlay file system also serves the purpose of mitigating the attack. As soon as the model detects malicious activity, the overlay file system initiates a moving target defense strategy (MTD), which changes the name of a file after the file has been read by any process. This renders making any changes to the file impossible, which prevents the Ransomware from encrypting further user data. Additionally, the collected raw features are analyzed with respect to entropy of write operations as well as the number of reads and writes of different workloads. The file system operations for both malicious and benign workloads are put into perspective. The detection system has been deployed to a Raspberry Pi machine and has shown high performance in terms of speed of detection and accuracy of detection. Finally, the performance overhead of the detection system has been analyzed. With the detection system running, the speed of write operations has decreased two-fold in comparison to the machine running without the detection system.

Acknowledgments

I would like to express my gratitude to Dr. Alberto Huertas Celdrán, Jan von der Assen and Chao Feng for their continuous support and guidance throughout the writing of the thesis. I have learned a lot from their expertise and insights. I would like to thank Prof. Dr. Burkhard Stiller and the Communication Systems Group for giving me the opportunity to write my thesis on a topic which I was highly interested in. My studies at the University of Zurich have been a long run. I would like to thank my family and my friends for their patience and support during my studies.

Contents

Chapter 1

Introduction

In recent years, Internet of Things (IoT) devices have been growing in popularity and the numbers are constantly increasing. The number of devices is predicted to be 75 billion by 2025 [1]. IoT devices serve many purposes and have found applications in various fields, such as smart homes, health sector and smart cities [2]. Many IoT products were not designed with security in mind, as they were initially created as stand-alone devices, without access to the internet [3]. In addition, it is not trivial to invent an efficient malware detection framework, as such devices are heterogeneous in nature and are resource-constrained, hence the use of conventional protection mechanisms is limited. With the growing number of IoT devices, cyberattacks on IoT devices have become more prevalent. Noteworthy is the Mirai botnet which in 2016 took control over hundreds of thousands IoT devices and performed a distributed denial-of-service (DDoS) attack of over 600GBps [4]. The massive increase in the number of IoT devices requires better security measures, which need to be adapted so that the limited processing power of IoT devices is taken into consideration.

In recent years, Ransomware has become a prevalent attack vector and has caused financial losses across the globe. For instance, it is estimated that CryptoLocker has caused losses of around 42 millions USD [5]. Although CryptoLocker's target was the Windows operating system, with the rising significance of IoT devices, it is crucial to develop effective protection mechanisms against ransomware.

To respond to these threats, the detection is a key activity. Numerous approaches have been proposed to detect different families of malware, including Ransomware. Based on the entropy of files and file extensions, one can fairly accurately classify whether a file has been encrypted or otherwise. Using entropy as features for machine learning (ML) models, accuracy close to 100% was achieved [6]. However, there is no literature that would create a real-time detection system for Linux-based IoT devices that combines the classification of Ransomware and immediate mitigation techniques that would prevent any further harm to the device.

1.1 Description of Work

This thesis aims to create an end-to-end Ransomware detection and mitigation framework that is effectively deployed on Raspberry Pi devices. For collection of malicious and non-malicious behavior, this work uses the Filesystem in Userspace (FUSE) to create a virtual file system. Utilizing FUSE, the file system operations are being monitored and logged in real-time.

Since Ransomware is usually encrypting files greedily, the amount of file writes is expected to be high. Therefore, the number of file writes performed by processes is an important feature to keep track of. In case of write operation, additionally the entropy of data that is being written to a file is logged - if a file write has a high amount of randomness (i.e., high entropy close to 8), it is likely that the file write is containing encrypted information. In addition, the file extension of each file write/read are stored, as different file types have various entropy. For instance, a text file's entropy is on average below 5, whereas the entropy of images or compressed files in general is on average approaching 8.

For the simulation of benign workloads, the Raspberry Pi has been set up as a File Transfer Protocol (FTP) server. To simulate client behavior, Apache JMeter was used to simulate traffic. The file system operations have been collected to have a baseline, in order to further distinguish between malicious and non-malicious behavior. Three Linux-based Ransomware have been selected to collect malicious behavior, namely Ransomware-PoC, Roar and DarkRadiation. While each Ransomware was running, JMeter was also initiated to simulate real-world settings, where the Ransomware will encrypt data in parallel while serving FTP requests.

The collected data was split up into time periods (2s, 5s, 10s), where in each time period the data has been aggregated so that in the end the sum of all writes and reads in that period is obtained. On the other hand, in case of entropy, the minimum, maximum and average entropy for each time period was calculated. However, the statistics were calculated only based on file extensions that had average entropy below 5.5. Files that have on average high entropy were ignored, since this would not provide any useful information. In the end, for each time window there was one aggregated row which was then fed into ML models.

Random forest classifier has shown highest accuracy for both benign and malicious workloads, reaching accuracy close to 100%. Logistic regression was inaccurate (30% accuracy) in case of testing on unseen Ransomware, specifically DarkRadiation. Finally, isolation forest was trained on benign (non-anomalous) dataset and has produced high accuracy rate in case of malicious behavior (close to 100% in case of Ransomware PoC and DarkRadiation). However, the benign dataset has been classified correctly only in 80% of cases, which implies high false positive rate.

Finally, MTD was applied after the behavior has been classified as malicious. Namely, after each file read the name of the read file is changed, so that the Ransomware is unable to write to the file after read. Additionally, a design of a buffering strategy has been proposed. For a given time period window (2s, 5s, 10s), files are buffered in memory until the next batch of file system operations is evaluated as malicious or non-malicious. In the

case of non-malicious classification, buffered data is finally stored into memory, otherwise the original, non-edited version of the file is retained.

1.2 Thesis Outline

In Chapter 2, the significance of Ransomware is introduced, including the financial harm caused by Ransomware in the past. Further, an introduction into supervised and unsupervised machine learning is provided. MTD paradigm is briefly explained as well as FUSE which was utilized in this thesis to implement MTD on file system level.

Chapter 3 gives an overview over related work on which this thesis builds. Different types of Ransomware detection approaches from previous works are shown. The chapter shows MTD techniques used for the mitigation of Ransomware. Finally, previous works are put in relation with this thesis and the contribution of this thesis is highlighted.

After that, Chapter 4 shows the files obtained from data corpus which were used as target to the different Ransomware samples. The methodology of data collection that was used for model training is shown. The features extracted from Ransomware behavior are and their relevance are introduced.

Further, Chapter 5 gives an overview over the Ransomware samples used in this thesis. Additionally, the Chapter introduces the MTD design, its architecture, as well as the data cleaning and data aggregation. For the training of models, non-malicious data is needed. The chapter shows the methodology of obtaining the benign dataset. Finally, the interaction between the detection system and the overlay file system and the final architecture which was used is shown.

Chapter 6 shows the results of this thesis. The raw data collected that was obtained by logging the file system operation is analyzed. Further, the performance of each model considered is shown. Additionally, performance overhead and speed of detection of the deployed system is shown.

Finally, Chapter 7 summarizes this thesis and gives further directions which future works could follow, which include improving the robustness of the model and prevention of further file encryption using buffering. Main limitations of the detection system are discussed.

Chapter 2

Background

For this thesis, the behavioral aspects of Ransomware and the Moving Target Defense (MTD) paradigm are critical. Therefore, they are introduced to the reader in the following sections. In addition, since substantial part of this thesis is detection and classification of Ransomware, definitions of supervised and unsupervised machine learning algorithms, as well as concrete algorithms that were used are introduced.

2.1 Ransomware

Ransomware is a type of malware which limits victims' of access to their machine by encrypting victims' valuable files and asks for ransom for decryption, whereas the attacker is the only person that holds the decryption key with which the encrypted data can be decrypted. Only after the ransom has been paid, the attacker may consider restoring encrypted data.

The usual payment method of ransom is Bitcoin [7] which offers the attackers pseudo-anonymity. It has been estimated that in a 2-year period, 16 million USD have been tracked to Bitcoin addresses that were likely receiving ransoms on BTC-e platform [8].

Well-known example of this kind of Ransomware is WannaCry, which was responsible for one of the largest attacks in history and caused harm to devices that were running on Windows operating system. Among others, it was reported that several hospitals, infecting and encrypting machines such as MRI devices [9].

Noteworthy is also non-encrypting Ransomware, also called locker-ransomware. Such Ransomware is locking user's machine and demanding ransom in return for unlocking the machine. Examples of this type of Ransomware are Reveton and Winlocker.[10]

Dynamic and Static Analysis are two main approaches for detecting Ransomware, as well as other types of malware. As Encrypting Ransomware are actively encrypting user data, classification algorithms can be utilized to detect whether there is a process running that behaves differently compared to other non-malicious processes. The operations that

Ransomware performs during runtime can be used as features to train and deploy a machine learning model that will detect Ransomware.

2.2 Machine Learning Techniques

Classification algorithms are a type of machine learning algorithms that, based on some data points, attempt to predict their class. Such machine learning algorithms have found usage in many different applications. For instance, in medicine, predicting cardiovascular diseases [11]. Unsurprisingly, classification algorithms have also found usage in security world [12], where the main goal of such algorithms is to predict malicious and non-malicious software or activity based on various features.

In contrast to anomaly detection, classification algorithms belong under supervised learning. In case of a classification task, the machine learning algorithm makes predictions of data points belonging to a set of pre-defined classes. Ransomware detection can be used in the context of a binary classification framework, where each data point is either malicious or non-malicious and the goal is for the algorithm to correctly predict the class of each sample.

2.2.1 Supervised Machine Learning

Supervised machine learning is a type of machine learning where the algorithm is trained on a labelled dataset. This means that during training phase, in addition to each sample's features, also the correct class of the data point is fed to the model.

Example: Decision Tree

A decision tree is a tree-like structure consisting of nodes and edges. Each node represents a decision, based on which the data is further partitioned. Starting at the root node, the tree is traversed until a leaf of the tree is reached, where the leaf represents the predicted label in case of classification. In 2.1 an example of a decision tree is depicted. One of the advantages of decision trees is that they can be easily interpreted - the tree splits and finally, upon reaching a leaf, the predicted label is obtained. Decision trees may be used both for classification and regression tasks. Decision tree is a basis for other more advanced ML algorithms, such as random forest classifier, which has been used for Ransomware detection in this thesis.

2.2.2 Unsupervised Machine Learning

Unlike supervised learning algorithms, unsupervised algorithms are not provided the labels and should instead actively be able to separate between classes based on patterns during

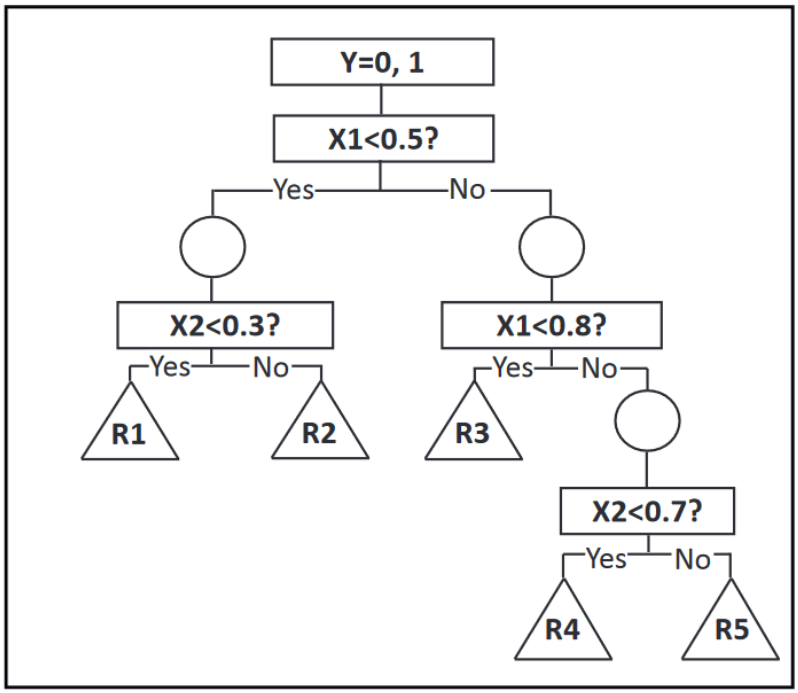

Figure 2.1: Decision Tree [13]

training phase [14]. Such algorithms are especially useful when having dataset without explicitly defined labels.

Anomaly detection is a technique used to identify instances that deviate from standard behavior or patterns. Typically, Anomaly Detection techniques belong to unsupervised machine learning. In ransomware detection, anomaly detection algorithms look for behavior or characteristics that do not match the typical behavior of a legitimate process. During evaluation, samples that are malicious might be provided, and the model should correctly predict these samples as anomalies, since the samples do not conform with the usual behavior [15]. For anomaly detection to be successful, the malicious samples need to be sufficiently different from non-malicious samples, so that the model can correctly discriminate between both. There exist both supervised and unsupervised machine learning algorithms for Anomaly Detection.

2.3 Moving Target Defense

The ideology behind MTD (Moving Target Defense) is that it is virtually impossible to have perfect protection against malware. MTD is a mechanism that keeps changing the system to reduce the time window of potential attacks [16]. The aim is to make eventual attacks much more difficult, so that the amount of harm caused is minimized. Three main elements of MTD have been described by [17]:

- What to move: there are multiple attack surfaces through which the attacker may

compromise the system. Each attack surface consists of parameters that are to be changed (i.e. moved) to evade the attack. Examples of parameters include software, hardware, IP address or the operating system itself

- How to move: which parameter of the attack surface should be moved. The parameter of the attack surface is moved either randomly [18], or based on behavior of the attack [19].
- When to move: defines when to move the parameter of the attack surface as well as the frequency of the move.

2.4 Entropy with Relation to Encrypted Files

Entropy is measuring randomness. There are different ways how Entropy can be calculated, but one of the commonly used Entropy is Shannon Entropy, which is defined by the following formula

$$H = -K \sum_{i=1}^{m} p_i log(p_i) \tag{2.1}$$

Since encrypted data appears to be random, entropy value of a file is a useful metric in detecting whether some data have been encrypted or not. However, since there exist file types that have high entropy by default, entropy alone is not sufficient for detection of encrypted files. Thus, one approach for encryption detection is comparing the entropy of a specific file to the average entropy of its file type.

2.5 Filesystem in Userspace

FUSE is a file system running in user-space. Using FUSE, the development of custom file system is less complex compared to creation of file system in kernel, since FUSE provides a simple API for modifications of file system. FUSE consists of two parts: kernel part and user-level daemon. After a request to file system, FUSE's daemon then processes the request and returns the control back to the VFS (Virtual Filesystem) [20]. 2.2 shows a high-level architecture of FUSE.

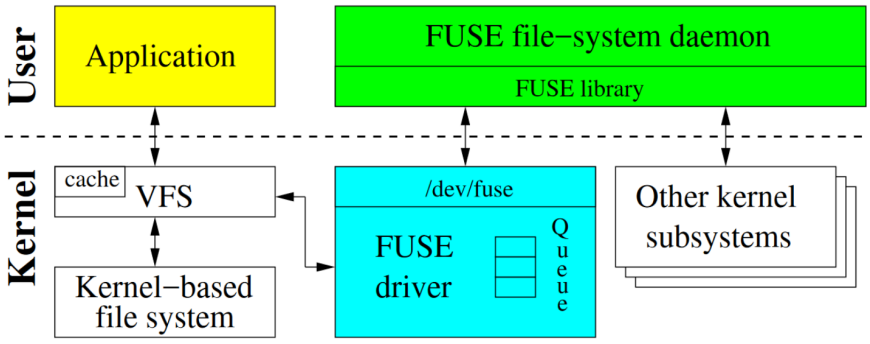

Figure 2.2: FUSE architecture [20]

Chapter 3

Related Work

This chapter introduces relevant literature that served as a basis for this thesis, ranging from detection mechanisms for Ransomware through analysis of relevant behavior to monitor, up to Ransomware mitigation techniques. Finally, an overview table is provided where previous works are summarized and the contribution of this thesis is highlighted.

3.1 Ransomware Detection

Cryptolock [21] aims to protect against Windows-based Ransomware by raising a warning to the user that an encryption might be taking place on the system. Every process is scored by a reputation score that indicates how suspicious the process is over time. The reputation score is calculated based on features such as Shannon Entropy that indicates how much randomness is present in a file. Similarity between the original version of the file and the updated version after write is measured using a similarity hash function sdhash [22]. Instead of collecting the behavior through monitoring of API calls, the reputation score is calculated after a file write has already been performed.

Redemption [23] intercepts every write and instead redirects the write operation to a "reflected" file, which is a copy of the original file. The original file is thus preserved and in case of suspicious activity the user is notified, and the original files can be restored. This was made possible by implementing a kernel that is able to intercept file system operations. Entropy ratio between the new and old data blocks are calculated. High ratio might indicate that encryption is taking place. Since Ransomware usually overwrites the data with an encrypted content, File Content Overwrite is used as an additional feature and represents the amount of data that is being overwritten with relation to the original file. Additionally, time aspect is also considered by calculating access frequency, which is a time window between two file write calls requested by the same process. Those features are then aggregated and a "malice score" is calculated for each process. Upon reaching a certain threshold, the process is flagged and reported to the user.

A detection system called RansomSpector [24] monitors both file system as well as network activities in order to provide more accurate results. The system is split into two parts,

namely Monitor and Detector. Monitor captures the system calls via Hypervisor and checks whether the system call is related to file system or network operation. If it is, the system call is then sent to the Detector which then performs pattern matching and if both file system and network operations match the malicious pattern, the process is identified as malicious. Otherwise, if there is only file pattern match and the write entropy is above certain threshold, the user is notified about suspicious activity. In the case the system call is neither file system nor network related, it is returned to the guest operating system and is executed normally. RansomSpector has been evaluated on Windows 7.

ShieldFS [25] For the data collection, I/O file system sniffer has been developed that in addition to raw IRP (I/O request packets) also adds additional information, namely entropy, process identifier (PID) and timestamp. The main features collected are write entropy, number of files renamed, number of files written, folder listing and number of files read. ShieldFS splits data accessed by a process into intervals called Ticks. Tick represents a proportion of files that have been accessed in relation to all files present. For instance, one Tick may represent 2% of all files. For every Tick, a classifier is evaluated on this subsection of file operations. In addition, there is a hierarchy of Ticks (e.g., 2%, 5%, 10%) and there is a separate classifier for each level of Tick. This ensures that the long-term behavior of a process is monitored as well. ShieldFS can recover the encrypted files in the case of malicious activity, since the original version of files is copied.

[6] have used different measures of entropy to mitigate the problem that various file types have various distribution of entropy value. The results of the calculated entropy are used as features for classification of encrypted files. On those features, machine learning models such as KNN, Decision Tree, Neural Networks were trained. The paper has not been deployed and tested on real-world machines.

3.2 Moving Target Defense Mechanisms for Ransomware

[26] has proposed a framework of MTD mechanisms which aim to mitigate Ransomware attacks and prevent data encryption on Raspberry Pi devices. The thesis makes use of FUSE to make adjustments to file system behavior. In total, five MTD mechanisms were proposed, out of which three of them were implemented.

Infinite Directory Depth MTD is a design where on every directory listing, a new empty directory is created. This causes the Ransomware to infinitely traverse through directories with no content and thus, under the assumption that the Ransomware enters directories in alphabetical order, no encryption takes place.

Given that some Ransomware may only encrypt specific file types, File Type Identification Change MTD changes the file extension so that the Ransomware does not encrypt the file with a different file extension, as the file is of no interest for encryption purposes.

The final MTD that was implemented is reducing read and write speed. By itself, this does not fully protect against Ransomware, but the amount of files that will be encrypted is significantly reduced.

Table 3.1: Categorization of Surveyed Related Work

Paper	*Mitigation*	*Detection*	*OS*	*Entropy*	*Evaluation*	*Data Collection*
[21]	user notification, process termination	yes	Windows	yes	yes user data	changes in
[6]	none	yes	none	yes	no	none
[24]	user notification	yes	Windows	yes	yes	system calls,
[24]						network traffic
[23]	process termination	yes	Windows	yes	yes	I/O system
[25]	discard writes	yes	Windows	yes	yes	IRPLogger
[26]	MTD techniques	none	Debian	no	yes	none
This	filename change	yes	Raspbian	yes	yes	FS interception

The first MTD that was proposed but not yet implemented is File Identification Change. After every file read, the file name is changed so that the Ransomware is unable to write to the file, since the file with the associated file name that was read is no longer existent.

Directory Name Change is the second MTD that is yet to be implemented. Since some Ransomware actively searches for specific directories, the proposed design is to change the directory name, so that the directories relevant for the Ransomware will no longer be present.

3.3 Discussion

Substantial amount of research has been done on detection of Ransomware on file system level. Most of the aforementioned papers use entropy as a possible indicator of encrypting behavior. However, to our best knowledge, no related works designed a system that would collect in FUSE. [25] have created a sniffer which captures IRPs (I/O request packets) and enriches them with additional metadata, such as entropy. [24], on the other hand, used hypervisor to capture system calls to watch file system operations. Most of the techniques that were evaluated on real-world machines were running on Windows operating system. No previous work has been done that would combine Ransomware detection and mitigation and finally evaluate the performance and feasibility on resource constrained IoT devices, such as Raspberry Pi.

In addition, most of the mitigation techniques were either terminating the malicious process or notifying the user. Dynamic mitigation after malicious activity detection, such as changing filenames to prevent encryption, has not yet been implemented. In this thesis, an end-to-end detection and mitigation system on Raspberry Pi was proposed and implemented. For the mitigation part, file name change MTD has been implemented, as proposed in [26]. In Table 3.1 the related work is summarized.

Chapter 4

Data

In this chapter the data corpus and its content which was used throughout the thesis is introduced. The methodology on how the file system operations have been collected from within the FUSE and the different features that have been extracted is shown.

4.1 Data Corpus

To collect the behavior of Ransomware, a set of files have been collected from digitalcorpora.org [27]. The collection contains large amount of various type of files, including images, text files, compressed files which makes it ideal for our use case. In total, around 20GB of files were downloaded and used as a target for encryption on the Raspberry Pi device.

4.2 FUSE Data Collection

The FUSE Go library [28] has been used to create a file system that would be able to collect file system related operations in real-time. The logs of file system operations were used both in training of the models and in evaluation.

The aim was to store the file system operations to a file which can then be further analyzed. The following information is logged:

- File writes - as Ransomware tends to access and overwrite files greedily, file writes are an important factor in determining if the activity is malicious or non-malicious
- File reads - similarly to file writes, high number of reads is an indicator of Ransomware greedily accessing files
- Entropy of file writes - for every file write operation, the corresponding entropy of the data that is being written to a file is recorded. High entropy is an indicator of possible encryption taking place

- PID - every file write/read is associated with the corresponding PID of the process that requested the operation. This makes it possible to differentiate between malicious and benign processes
- Timestamp - delta in seconds from the initial mount of FUSE and the file system operation. Timestamp is used to aggregate the records into time windows at a later point. For example, given a time window of 5 seconds, records with timestamps 1, 2, 3, 4 will be bucketed into a single time window.
- File Extension - different file types have different average entropy. Thus, it is important to differentiate the file type that is being written to. For instance, the write entropy to compressed file types is expected to be high, since those file types have high entropy by default.

For raw data collection, the overlay file system is structured as shown in Figure 4.2. The root folder of the overlay file system has two children, namely "Malicious Folder" and "Benign Folder". The files from data corpus are stored in the malicious folder, which is also the target for the Ransomware. Folders within the malicious folder are structured based on names of the zip files downloaded from the data corpus. The benign folder is where the files from FTP traffic are being uploaded and downloaded. Since FTP is also actively listing the directory and downloading files, the goal of this layout is to not contaminate the FTP traffic with encrypted files. Otherwise, the encrypted files could circulate in the FTP traffic and the benign dataset would no longer be representative of benign FTP workload. However, this structure is only relevant for obtaining the training datasets. As soon as the trained model is deployed, even if there are encrypted files circulating in the FTP, high entropy would indicate that the file has been encrypted at some point in time in the past.

The file type of the log is a CSV and has a schema as depicted in Figure 4.1. All the features were collected after the interception of the file system operation in the overlay system. Finally, after logging of the specific file system operation, FUSE returns the control to the kernel, which then writes the data on disk. For training purposes, only one CSV file with logs has been created for the whole data collection period. However, for evaluation, a new CSV file is generated for every time window which is then consumed by the detection system.

In addition, the number of writes and reads were aggregated for each time window and PID. Ransomware usually encrypts files greedily, therefore a high number of reads and writes is to be expected.

pid	entropy	op	ext	timestamp
3680	-1.000000	read	html	12
3680	-1.000000	read	html	12
3680	-1.000000	read	html	12
3680	-1.000000	read	html	12
3680	-1.000000	read	log	12
3680	-1.000000	read	log	12
3680	-1.000000	read	html	12
3680	-1.000000	read	html	12
3680	-1.000000	read	unk	12

Figure 4.1: Collected Data

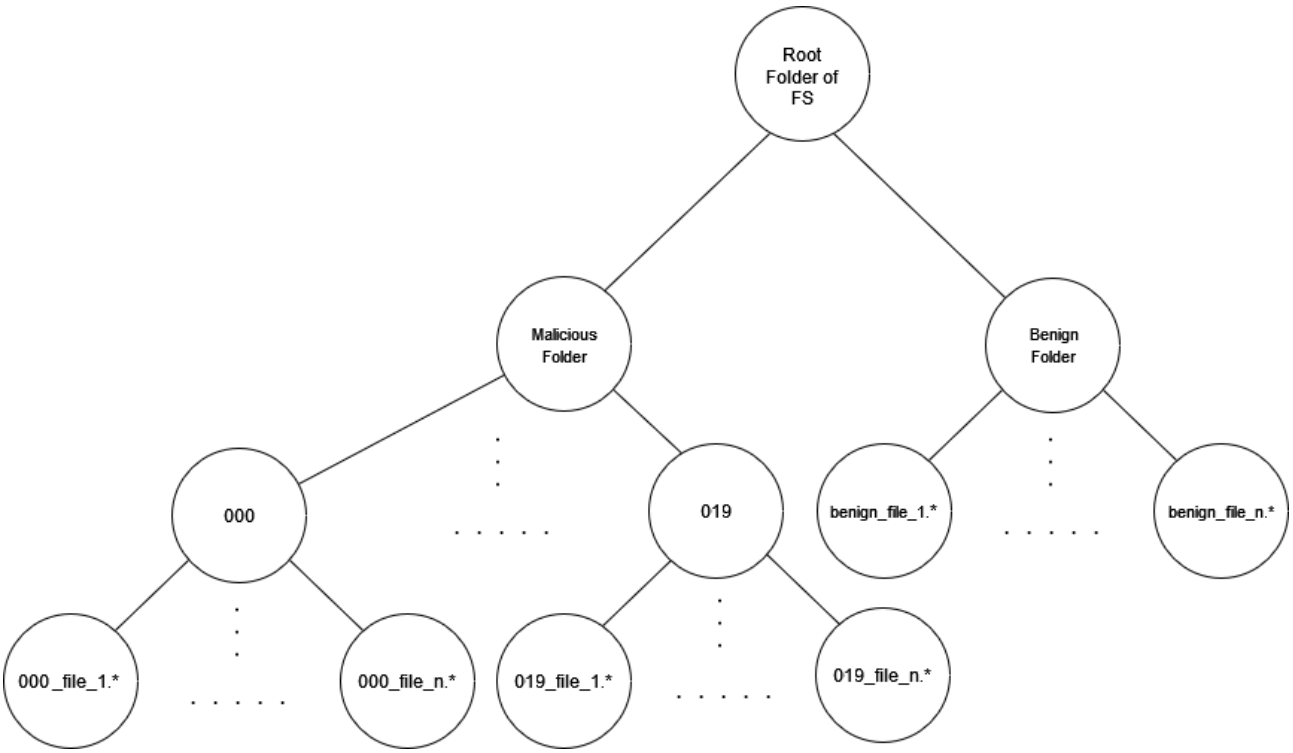

Figure 4.2: Overlay file system architecture

Chapter 5

System Design

In this chapter, the setup that has been used in this thesis with regard to hardware is introduced. Additionally, the methodology of detection system is introduced and explained how the MTD interacts with the detection system.

5.1 Setup

A Raspberry Pi 4B with 2 GB of memory has been used with Raspberry Pi operating system installed. An FTP server has been set up on the device and continuous FTP traffic consisting of files obtained from (DL or also upload?) DigitalCorpora.org [27] is present on the device to simulate benign behavior. The FTP server scenario was chosen since frequent FTP traffic is arguably the closest behavior resembling Ransomware, since FTP traffic consists of frequent file writes and reads. There is a dedicated folder for FTP on the device where the files are being uploaded and downloaded via FTP. In addition, this folder also contains around 20 GB of files from DigitalCorpora.org [27]. The overlay file system is mounted on the mentioned FTP folder and is also a target to the various Ransomware's samples.

The FUSE Go library [28] has been used to create a file system that would be able to collect file system related operations in real-time. The logs of file system operations were used both in training of the models and in evaluation. The second requirement is to have the FUSE setup such that MTD can be initialized if a file system activity has been flagged as malicious. For the model training, Python programming language has been used, utilizing the scikit-learn library.

5.2 Ransomware Samples

In this thesis, three Linux-based Ransomware samples have been selected, on which the data has been collected and models trained. More specifically, DarkRadiation, Ransomware PoC and Roar.

- Ransomware PoC: Ransomware that is written in Python programming language. When running the python script, user can specify which directory is to be encrypted by changing the corresponding argument. Ransomware PoC is changing the file extension after a file has been encrypted to .wasted.
- DarkRadiation: This Ransomware is written purely in shell scripts. DarkRadiation encrypts home directory by default, but in our use-case this was changed to only encrypt the corresponding root folder of the overlay file system. Similarly to Ransomware PoC, DarkRadiation changed the file extension after encryption.
- Roar: Similar to Ransomware PoC, Roar has been written in Python. Roar has been specifically designed to evade dynamic detection by utilizing Reinforcement Learning. Compared to the other two Ransomware samples, Roar's encryption rate is considerably slower. Roar changes the file extension to .wasted after encryption [29].

5.2.1 Benign Traffic Simulation

JMeter [30] has been used to simulate FTP traffic on the Raspberry Pi. Although JMeter is mainly used in the stress testing of servers domain, it fits the use-case, as peaks of traffic can be configured, as well as how many threads are actively creating the traffic on the FTP server, which makes the FTP traffic more realistic. The JMeter User Interface is depicted in Figure 5.1 and in Figure 5.2 the interface is shown where files are being actively uploaded to the FTP server. In Figure 5.3, the implementation of downloading files from the FTP server back to the user is shown. JMeter lists the directory and randomly chooses a file to download from the server. In addition to FTP upload, the goal is to also simulate the user listing through directories and downloading files. The JMeter client was active on a different machine, connecting to the FTP server via local network. All the file system operations caused by FTP traffic have been logged, and the collected benign data is further used during training of Ransomware detection models. In order to distinguish whether an operation was benign or malicious, every file that has been uploaded via FTP to Raspberry Pi has "benign" as a prefix in the file name, so that in the training phase can easily be distinguished and the corresponding label is added (1 for non-malicious and 0 for malicious) based on the filename on which the operation was performed.

The JMeter configuration was set up as follows:

- Files from DigitalCorpora.org [27] were residing on a local machine and were actively being uploaded to a Raspberry Pi folder with mounted FUSE. In this way, FUSE can log the operations. For this, two threads were used with a ramp-up period of 30 seconds.
- The content of the FTP directory was actively listed to get a list of all the available files. A random file has been chosen to download from a Raspberry Pi FTP folder back to the local machine. Again, 2 additional threads were dedicated to the downloading process.

The attempt was made to add more threads, but more than 2 threads caused the Raspberry Pi to freeze after a while and no further files were downloaded or uploaded. Thus, in total 4 threads for downloading and uploading was the maximum the Raspberry Pi was able to serve.

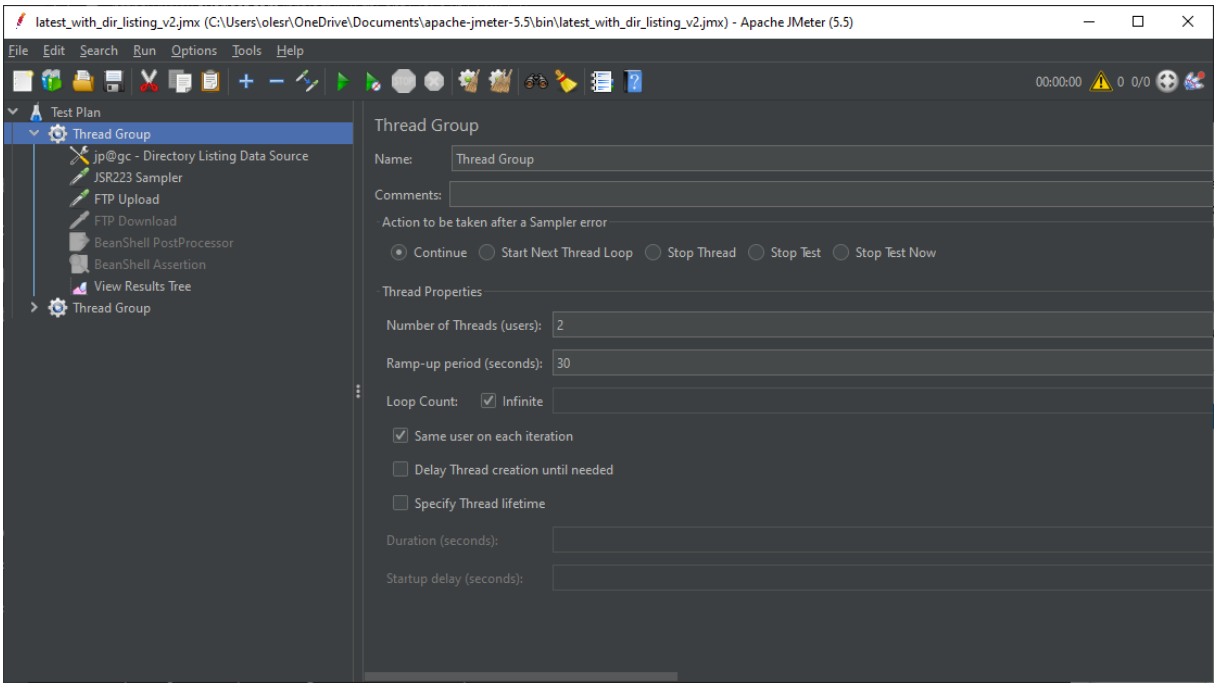

Figure 5.1: JMeter User Interface

5.2.2 Ransomware Behavior Collection

On an FTP server, Ransomware will hardly be the only process running in real world settings. Therefore, for training and evaluation of classification models, Ransomware was running inside the mounted folder concurrently with benign FTP traffic. Secondly, solely for data exploration and analysis, each Ransomware has been run separately without the benign traffic running in the background. The purpose was to have the Ransomware samples separated from any other input/output-heavy processes running in the background that would in any way affect the speed of reads, writes or the encryption in general. The target of the Ransomware was the mounted overlay file system root folder. In total, three different Linux-based Ransomware samples have been used to collect malicious activity. Each Ransomware has been running separately, so that in the end three datasets were obtained, where each dataset contains a combination of benign activity and specific Ransomware sample's activity.

5.3 Ransomware Detection System

After the collection of data via FUSE, two classification models and one anomaly detection model have been trained. This chapter introduces what pre-processing steps were required

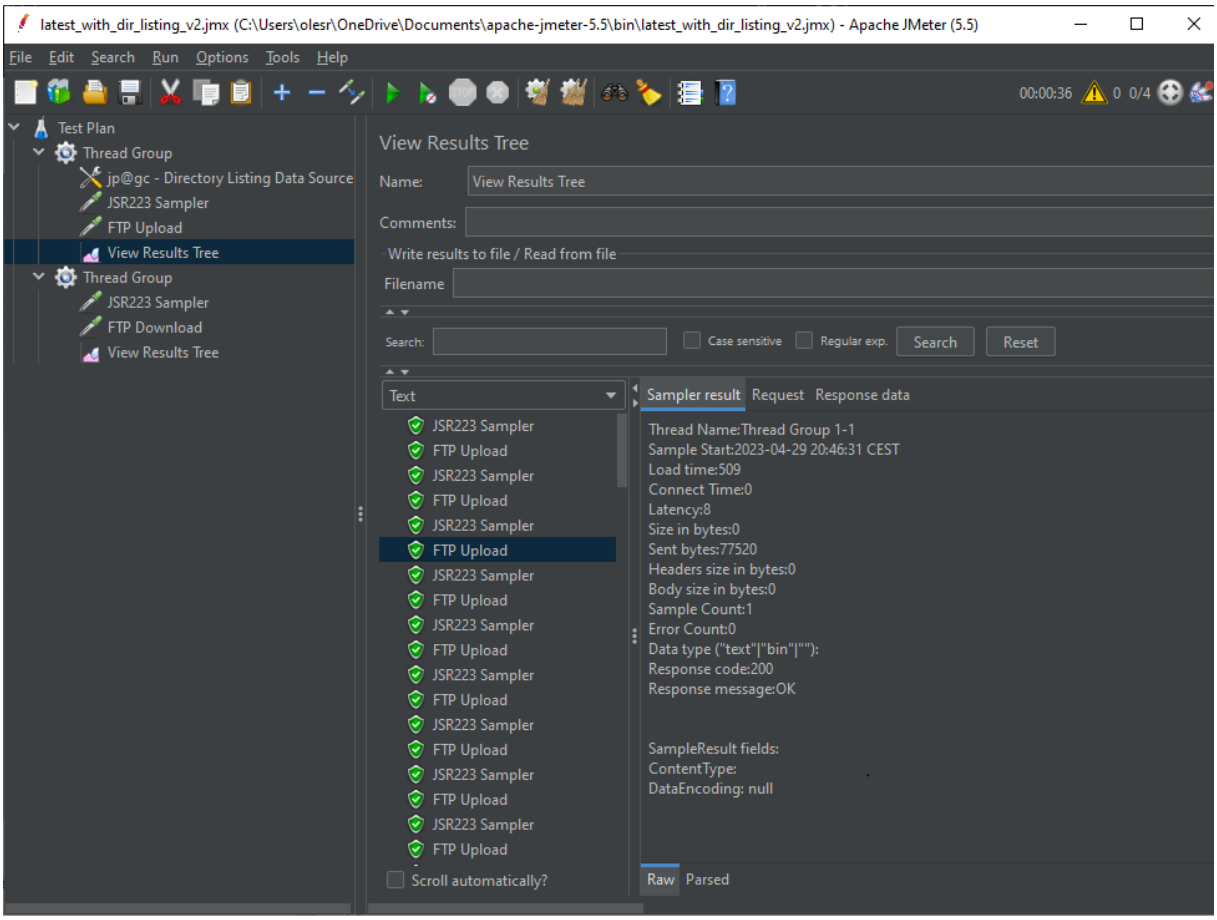

Figure 5.2: JMeter uploading to FTP server

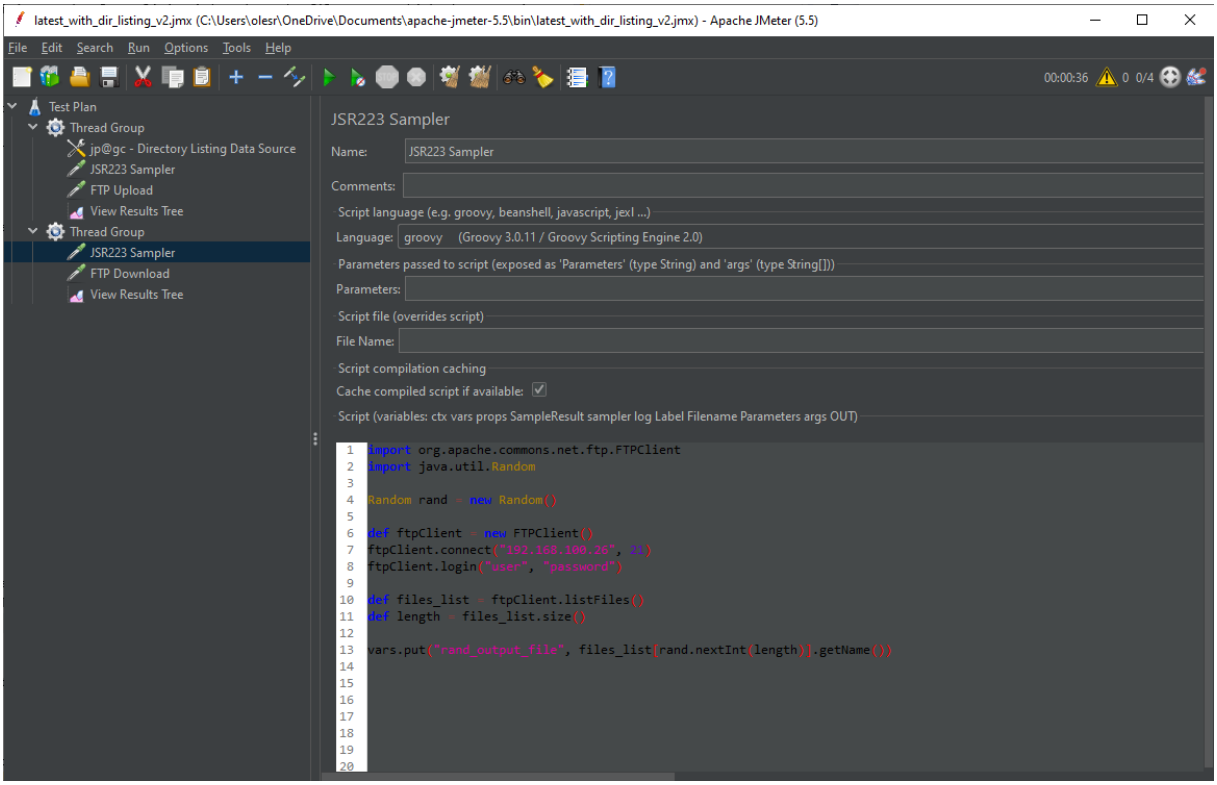

Figure 5.3: JMeter download from FTP server

and finally how the models have been trained.

5.3.1 Data Aggregation

As a part of pre-processing, the original CSV file with raw data has been aggregated by timestamp and by PID. Firstly, the timestamp was put into buckets, so that all operations that were performed within the same time window belong to the same bucket. In total there are three buckets, specifically 2, 5 and 10 seconds buckets. An illustration of this can be seen in Figure 5.4, where `time_id` column represents the corresponding bucket. The dataset was then grouped by the ”time_id” column and ”pid” column. In the end, a dataset was generated where there is a separate record for the operations performed by each process in a specific time window. For each record the number of writes have been aggregated and the number of reads as well as calculated aggregate statistics of write entropy, namely minimum entropy of write operation, maximum entropy of write operation and mean entropy of all write operations. The entropy has been calculated according to the Algorithm 1.

The statistics of entropy were calculated only on file types that have on average entropy below 5.5, as according to entropy analysis in evaluation (see Section 6.1.3) this threshold separated binary and compressed files well from the remaining low entropy file formats. The average entropy of each file type has been calculated based on collected data from benign traffic. The reasoning is as follows: if a process is writing to a file that has high entropy by default (e.g., jpg, zip) the write entropy will likely also be high, even though

Algorithm 1 Entropy Calculation Algorithm

$DataLength \leftarrow Length(DataBlock)$
$Entropy \leftarrow 0$
for $byte \leftarrow 1$ to 256 **do**
    $ByteCountMap[byte] \leftarrow 0$
end for
for $byte$ in $DataBlock$ **do**
    $ByteCountMap[byte] \leftarrow ByteCountMap[byte] + 1$
end for
for $byte$ in $GetKeys(ByteCountMap)$ **do**
    if $ByteCountMap[byte] == 0$ **then**
        $continue$
    end if
    $p \leftarrow ByteCountMap[byte]/DataLength$
    $Entropy \leftarrow Entropy - p * \log_2(p)$
end for
return $Entropy$

the operation is not malicious, and no encryption is taking place. If high entropy values would also be included in the entropy statistics calculation, aggregated entropy would be higher on average, which would make it more difficult to classify correctly as malicious or benign. Figure 5.4 shows the final structure of the aggregated data.

	time_id	sum_writes	sum_reads	pid	entropy_min	entropy_mean	entropy_max
0	20	1	1	20532	7.869912	7.869912	7.869912
1	20	2	1	20549	7.94428	7.951192	7.958104
2	20	2	1	20693	7.890738	7.9195205	7.948303
3	20	4	1	20546	7.947358	7.9522965	7.956804
4	20	7	2	20533	7.947251	7.952903	7.960178

Figure 5.4: Data Aggregated by Time Window and PID

5.3.2 Training and Model Deployment

The aggregated data has been used to train random forest classifier [31], logistic regression [32] and isolation forest [33]. There were two approaches to model training for the mentioned classification models. The first approach was to train the models on data collected from all three Ransomware as well as benign traffic. The data was split in a way where 80% of the data was used for training and 20% for testing the trained model. As all models have shown high accuracy, there was no need to have validation split for parameter tuning. The second approach was to train the models only on 2 Ransomware datasets

and benign dataset to test the accuracy on behavior of unseen Ransomware sample. This exercise of leaving out one Ransomware sample was done on all three Ransomware samples, so that in the end, for each time period, there were three models, each trained on 2 Ransomware samples. Both of the approaches were repeated for all data grouped by different time window, namely 2, 5 and 10 seconds.

The same aggregated data has been used to train the isolation forest anomaly detection model. Different to classification models, isolation forest has been trained only on benign dataset and based on that the model should correctly predict whether the data sample is anomalous (malicious) or non-anomalous (benign). Similarly to Classification models, isolation forest has been trained on 2, 5 and 10 seconds time windows.

The models have been trained and evaluated on a local machine on the collected datasets. For deployment on the Raspberry Pi device, the random forest classifier model has been used, as in both approaches the random forest classifier has outperformed both logistic regression and isolation forest in terms of detection accuracy (false positive as well as false negative). The deployed model has been trained on a 5 seconds aggregation level, since the model provides higher accuracy on unseen Ransomware samples compared to 2 seconds time window, as well as providing potentially faster response compared to 10 seconds aggregation level and hence less user data will be encrypted in the 5 seconds time period. The deployed model that has been trained according to the first approach, where all 3 Ransomware samples have been used for training.

On the Raspberry Pi machine, the model is running in an endless loop and evaluating the CSV files outputted by FUSE. As soon as the model evaluates some data sample as malicious, the endless loop of evaluation is terminated and the process writes ”true” into a separate log file, which indicates that there is malicious activity in the file system. Otherwise, the log file contains ”false” standing for no malicious activity detected by the model (see Algorithm 2). The log file is then read by overlay file system whenever there is a write/read request to determine whether the MTD is to be initiated or to perform the write/read operation in the standard way.

Algorithm 2 Detection System Algorithm

```
    while True do
2:      if log.csv exists then
            PreProcessedLog ← PreProcess(log.csv)
4:          Predictions ← Predict(PreProcessedLog)
            if malicious in Predictions then
6:              Write(”true”, classifier.log)
                break
8:          end if
        else
10:         Wait(time window)
        end if
12: end while
```

5.4 MTD Implementation

The file name change MTD technique proposed by [26] has been implemented in this work. After a file is read, the file name of the file is changed, so that the process that is attempting to write to that file will be unable to do so, as the file name has been changed. The MTD is activated after the classification system has detected malicious activity. A prefix "_" to the original file name is added, so that the new file name is "_example.txt" instead of "example.txt". The workflow is illustrated in Figure 5.5.

In addition, buffering strategy (Algorithm 4) has been designed, however not implemented in this thesis. The data from write requests is buffered into memory, as long as there is no output from the model. As soon as the file into which the model writes changes, the overlay file system reads the file. If the file contains "true", the buffered data is dropped and the MTD is initiated. Otherwise, the buffered data is persisted to disk.

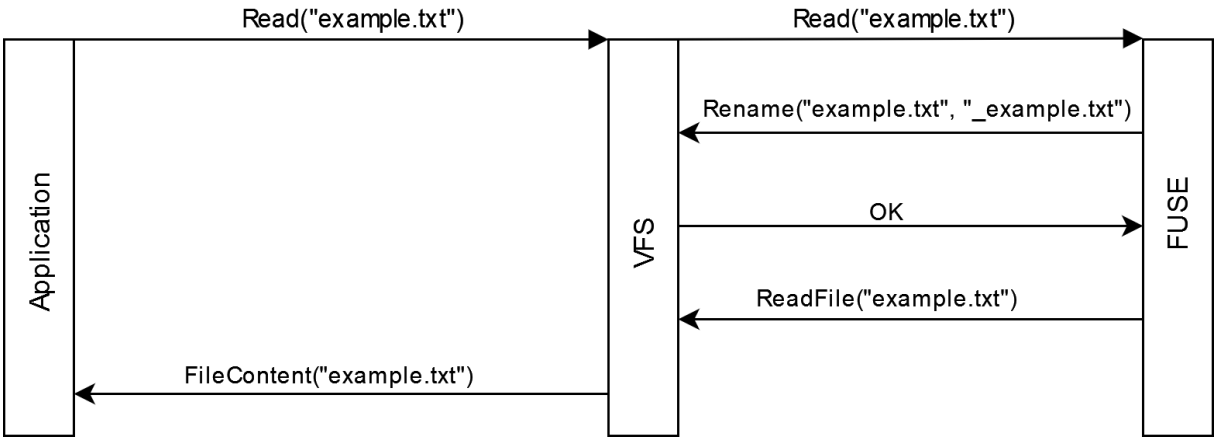

Figure 5.5: File Name Change MTD

5.5 Interaction between Detection System and FUSE

The final setup combines the logging functionality of FUSE, File Name Change MTD and detection system. Initially, the FUSE file system is activated, which collects file system operations into CSV logs. After every time period (time window specified by user), a new CSV file is created in which the logs are stored corresponding to that time span. Secondly, a model for detection that has been trained and deployed to Raspberry Pi machine is continuously watching the logs created by FUSE and classifies whether the activity based on the log file is malicious or not. If the model evaluates the activity as malicious, the flag "True" is written into a separate classifier.log file. In case of non-malicious activity, "False" is outputted. Then, the FUSE file system reads this file and as long as the flag is "False", the file system continues performing the standard activity - logging file system operations and handling requests. As soon as the model outputs "True", FUSE activates the File Name Change MTD, at which point no process will be able to write to the file system. The process is depicted in Figure 5.6.

Algorithm 3 MTD and Logging Algorithm

```
    while True do
        if FSRequest exists then
 3:         log.timestamp ← CurrentTime − InitTime
            log.pid ← FSrequest.pid
            if FSrequest.op == "write" then
 6:             log.op ← "write"
                log.entropy ← CalculateEntropy(FSrequest.data)
            end if
 9:         if FSrequest.op == "read" then
                log.op ← "read"
                log.entropy ← −1
12:         end if
            if "true" in classifier.log then
                rename(FSrequest.filename,"_" + FSrequest.filename)
15:         end if
            Write(log, log.csv))
        end if
18: end while
```

Algorithm 4 Buffering Algorithm

```
    while True do
        if FSRequest exists then
            if FSRequest == "write" then
 4:             Buffer.Enqueue(FSRequest.data)
            end if
        end if
        if HasChanged(classifier.log) then
 8:         if "true" in classifier.log then
                for _ in Buffer.length do
                    Buffer.dequeue()
                end for
12:             InitiateMTD()
            else
                for _ in Buffer.length do
                    Data ← Buffer.dequeue()
16:                 Write(Data)
                end for
            end if
        end if
20: end while
```

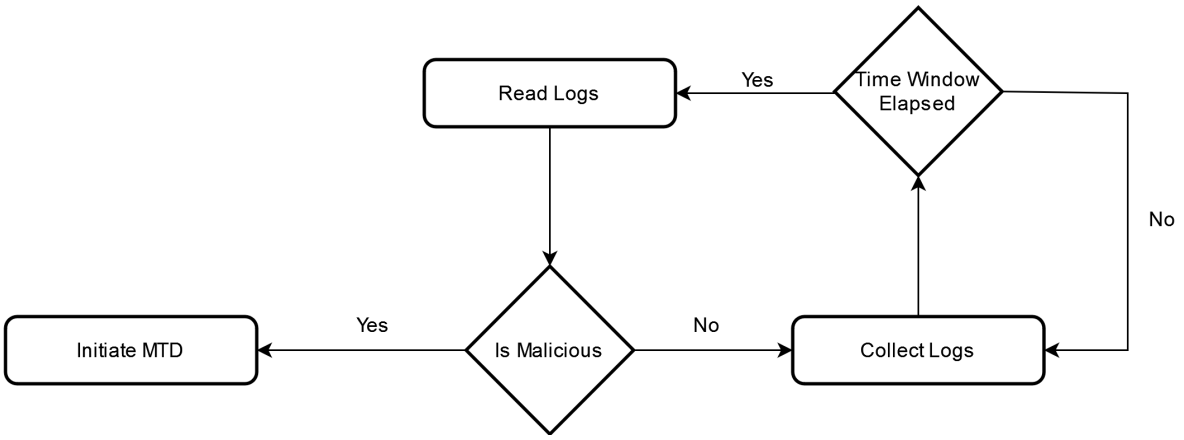

Figure 5.6: Detection System with MTD

Chapter 6

Evaluation

After implementing the detection system, the system was evaluated with regard to model classification accuracy. After deployment of the detection system to a Raspberry Pi machine, the speed of detection of different Ransomware samples has been measured. Apart from that, the number of encrypted files while the Ransomware sample is running undetected was measured. Finally, benchmarks of resource usage such as CPU usage and RAM have been monitored while the detection system is running and compared with the same machine without the detection system running.

6.1 Ransomware Behavior Analysis

In this section, the differences between benign workload and Ransomware are outlined and analyzed. This is especially important, since the accuracy of models depend on how well they can differentiate between the different data samples. Firstly, sample of the collected raw data for all three Ransomware are shown to observe differences in activity between them. Then the number of reads and writes of different workloads are visualized. Finally, entropy of write operations of Ransomware and benign workload is analyzed.

6.1.1 Raw Data

Raw data has been collected in a form of tabular format via the logging system implemented in FUSE. Before analyzing aggregated data, it is interesting to first look at sample raw data, as some Ransomware features may be observed that are no longer visible after aggregation. In the case of DarkRadiation (Figure 6.1), the most apparent trait is that there are different PIDs performing writes and reads, which indicates that DarkRadiation is spawning multiple processed for encryption. The write entropy is close to 8 for txt file extension. In addition, DarkRadiation is changing the file extension of all encrypted files. On the other hand, Ransomware PoC (Figure 6.2) has only one process running throughout the whole dataset. Similarly to DarkRadiation, Ransomware PoC is also performing high entropy writes to txt file type, however in addition to that, every high entropy write

operation also has a corresponding low entropy write, having the entropy value of around 4. Finally, Roar (Figure 6.3) exhibits similar behavior to Ransomware PoC, having only one process running throughout the observed period, as well as having alternating high and low entropy writes. The cause of this is likely the fact that both Ransomware PoC and Roar are using Python functionality to write encrypted content into files, whereas DarkRadiation is written in a shell script. Looking at the timestamp column (representing seconds since the initialization), the apparent difference is that the records are much less frequent in Roar compared to Ransomware PoC. This is expected, since Roar is encrypting files at a low encryption rate to evade detection [29]. Additionally, in figures 6.7 and 6.6 the average number of reads and writes per second respectively are visualized. DarkRadiation has the highest number of writes per second, followed by Ransomware PoC. On the other hand, FTP traffic performs relatively low amount of writes, possibly bottlenecked by network. Roar shows the lowest amount of writes per second. In all workloads, write operations are more frequent compared to read operations. Interestingly, Ransomware PoC shows the fastest read rate, followed by DarkRadiation, FTP traffic and Roar.

6.1.2 Amount of Reads/Writes of workloads

Three Ransomware samples have been selected for this thesis, namely Ransomware PoC, DarkRadiation and Roar. From the datasets obtained during data collection phase, the different number of writes and reads of each Ransomware as well as benign FTP traffic can be analyzed and compared. All three samples have shown different behavior with regard to file writes and reads. As can be seen in Figure 6.6, DarkRadiation Ransomware has the highest number of writes, followed by Ransomware PoC. The higher number of writes for DarkRadiation may be either due to multiprocessing (Figure 6.1), or a different encryption algorithm compared to Ransomware PoC. Roar on the other hand shows the lowest amount of file writes, also compared to the benign FTP traffic. Thus, file writes are clearly insufficient to separate malicious behavior from benign. Figure 6.7 shows the number of reads for each workflow. Number of reads and writes show similar patterns, namely if there is downfall of writes, decrease in reads can be observed and vice versa.

6.1.3 Entropy Analysis

Entropy is an important feature when training Ransomware classifiers, as entropy measures randomness in data, thus it is worth investigating the entropy of file writes of Ransomware as well as benign traffic. For the model training and evaluation, all file types that have on average entropy higher than 5.5 are disregarded and only the records with a file extension lower than 5.5 are fed into the models. In Figure 6.8 the file types with entropy below 5.5 threshold are displayed, which are mostly file extensions that store plain text data, such as csv, java and log. On the other hand, in Figure 6.9 all the remaining file extensions having entropy higher than the threshold are shown. The entropy of the files in the two Figures were calculated based on benign activity writing into the mounted file system.

pid	entropy	op	ext	filename	timestamp
3745	-1.000000	read	html	010990.html	109
3745	-1.000000	read	pdf	010105.pdf	109
3853	-1.000000	read	txt	010036.txt	109
3853	7.827150	write	txt	010036.txt.☢	109
3858	-1.000000	read	pdf	010205.pdf	109
3745	-1.000000	read	html	010967.html	109
3854	-1.000000	read	txt	010330.txt	109
3854	7.957054	write	txt	010330.txt.☢	109
3854	7.958537	write	txt	010330.txt.☢	109
3854	-1.000000	read	txt	010330.txt	109
3858	7.960800	write	pdf	010205.pdf.☢	109
3855	-1.000000	read	ps	010732.ps	109
3854	7.955325	write	txt	010330.txt.☢	109
3858	7.943759	write	pdf	010205.pdf.☢	109
3855	7.951780	write	ps	010732.ps.☢	109

Figure 6.1: Sample Data Collected From DarkRadiation

pid	entropy	op	ext	filename	timestamp
1406	-1.000000	read	txt	test.txt	6
1406	2.221252	write	txt	test.txt	6
1406	-1.000000	read	txt	010036.txt	7
1406	7.849031	write	txt	010036.txt	7
1406	1.584963	write	txt	010036.txt	7
1406	-1.000000	read	txt	010330.txt	7
1406	7.955077	write	txt	010330.txt	7
1406	3.875000	write	txt	010330.txt	7
1406	-1.000000	read	txt	010330.txt	7
1406	-1.000000	read	txt	010330.txt	7
1406	7.955981	write	txt	010330.txt	7
1406	4.000000	write	txt	010330.txt	7
1406	-1.000000	read	txt	010330.txt	7
1406	7.954020	write	txt	010330.txt	7
1406	4.000000	write	txt	010330.txt	7

Figure 6.2: Sample Data Collected From Ransomware PoC

pid	entropy	op	ext	filename	timestamp
19887	-1.000000	read	txt	010036.txt	28
19887	4.729030	write	txt	010036.txt	29
19887	1.584963	write	txt	010036.txt	29
19887	-1.000000	read	txt	010330.txt	29
19887	7.946878	write	txt	010330.txt	32
19887	3.875000	write	txt	010330.txt	32
19887	-1.000000	read	txt	010330.txt	32
19887	-1.000000	read	txt	010330.txt	32
19887	7.953166	write	txt	010330.txt	36
19887	3.750000	write	txt	010330.txt	36
19887	-1.000000	read	txt	010330.txt	36
19887	7.952701	write	txt	010330.txt	40
19887	4.000000	write	txt	010330.txt	40
19887	-1.000000	read	txt	010330.txt	40

Figure 6.3: Sample Data Collected From Roar

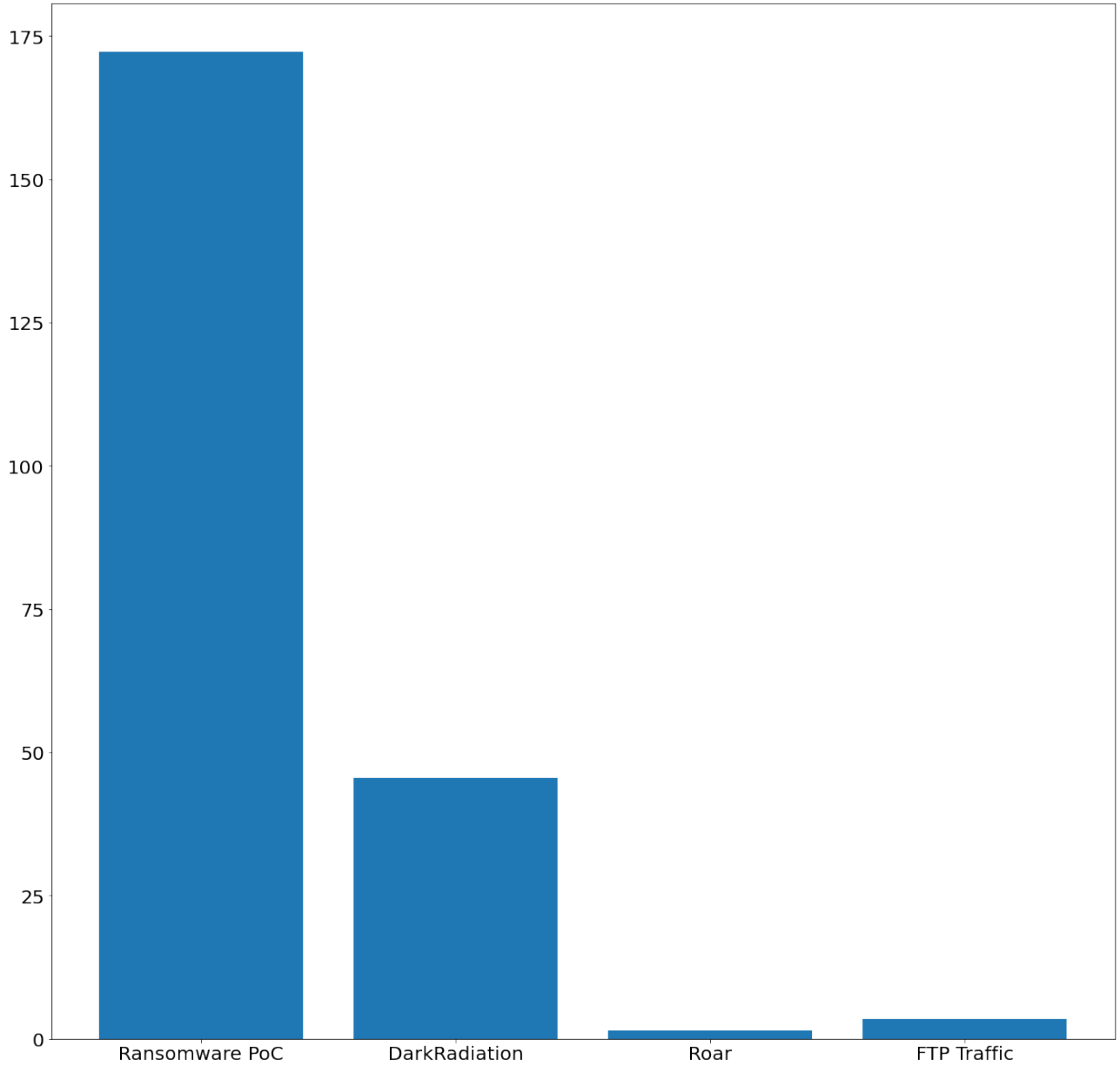

Figure 6.4: Average number of reads per second

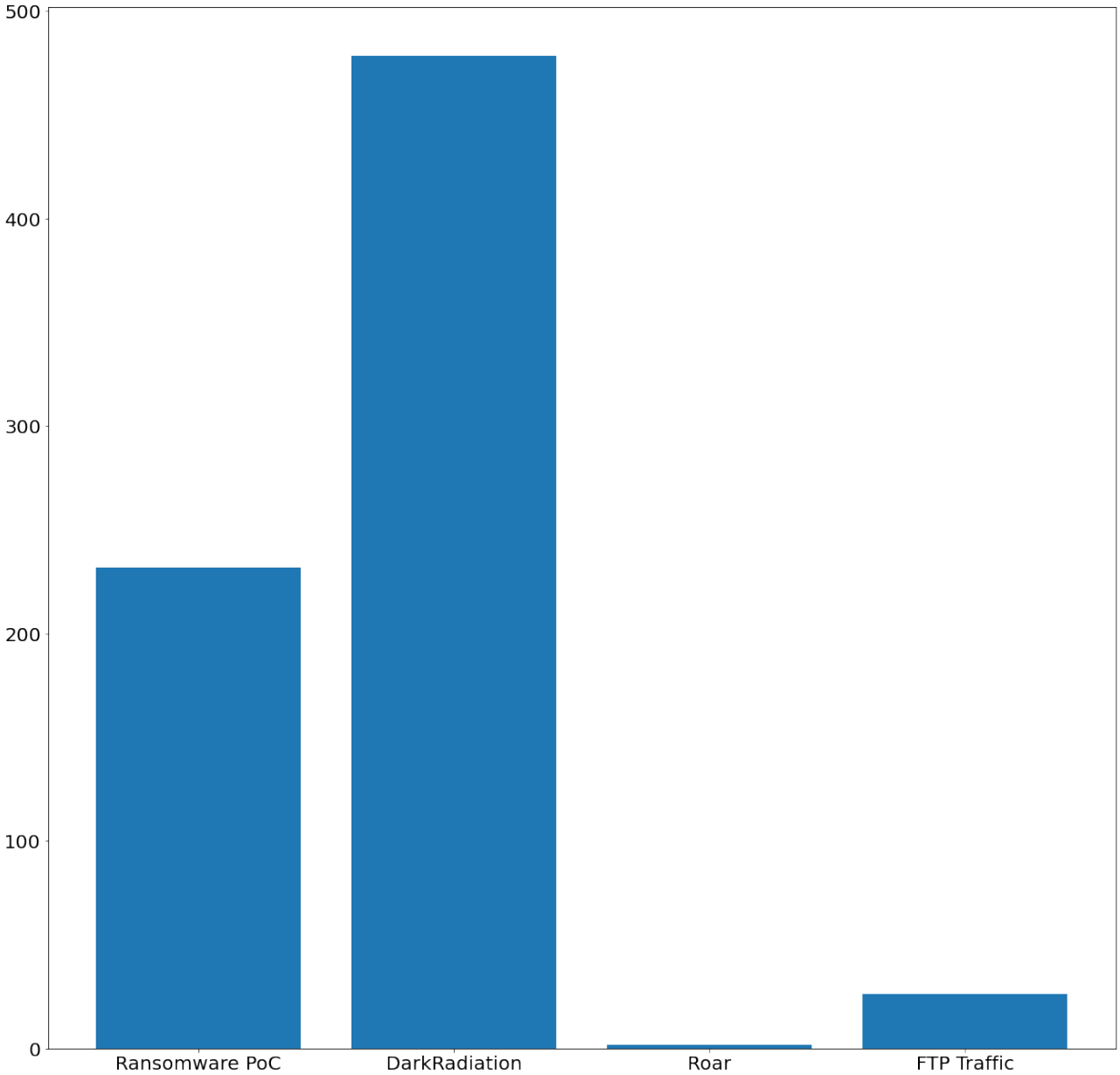

Figure 6.5: Average number of writes per second

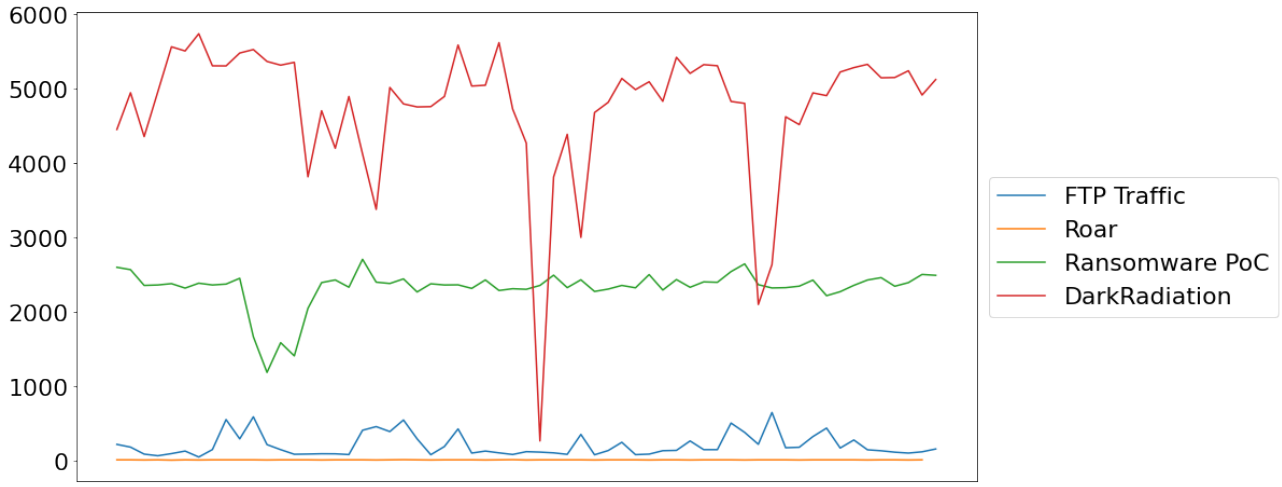

Figure 6.6: Number of Writes for each workload every 10 seconds for 10 minutes

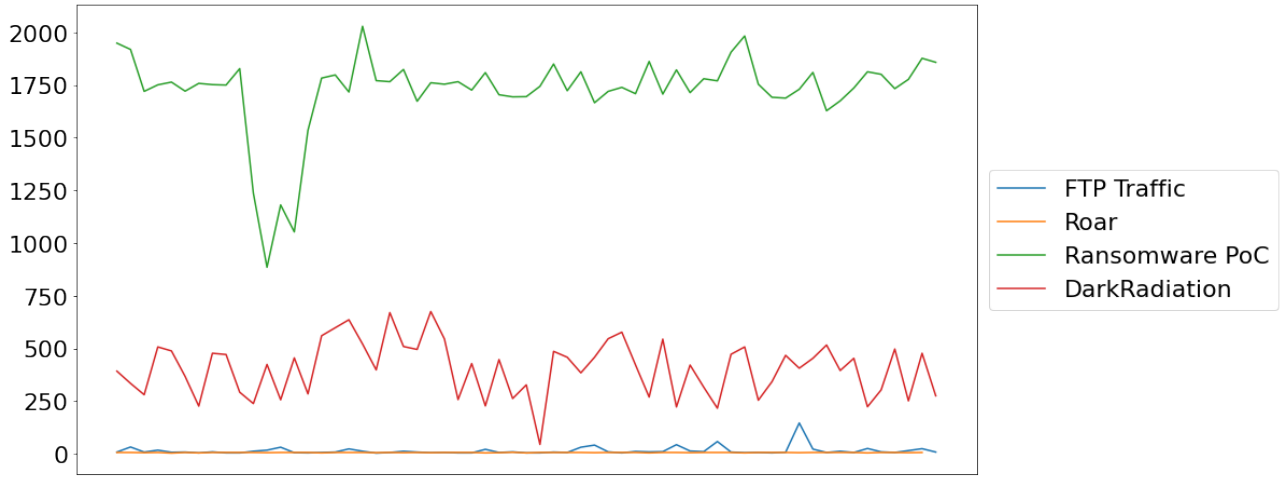

Figure 6.7: Number of Reads for each workload every 10 seconds for 10 minutes

In Figure 6.10 DarkRadiation's write entropy for different file types is depicted. For DarkRadiation, the entropy value is relatively constant at around 8 across all file types. On the other hand, Ransomware PoC and Roar the entropy is consistently around 6 for all extensions (Figure 6.11 and Figure 6.12). The Ransomware PoC and Roar average entropy is thus significantly lower compared to DarkRadiation. This behavior can be explained by looking at the raw data in Figure 6.2 and 6.3, where in addition to high entropy writes slightly below 8, both Ransomware PoC and Roar perform write operations with entropy of around 4. Averaging the entropy then results in entropy value of around 6.

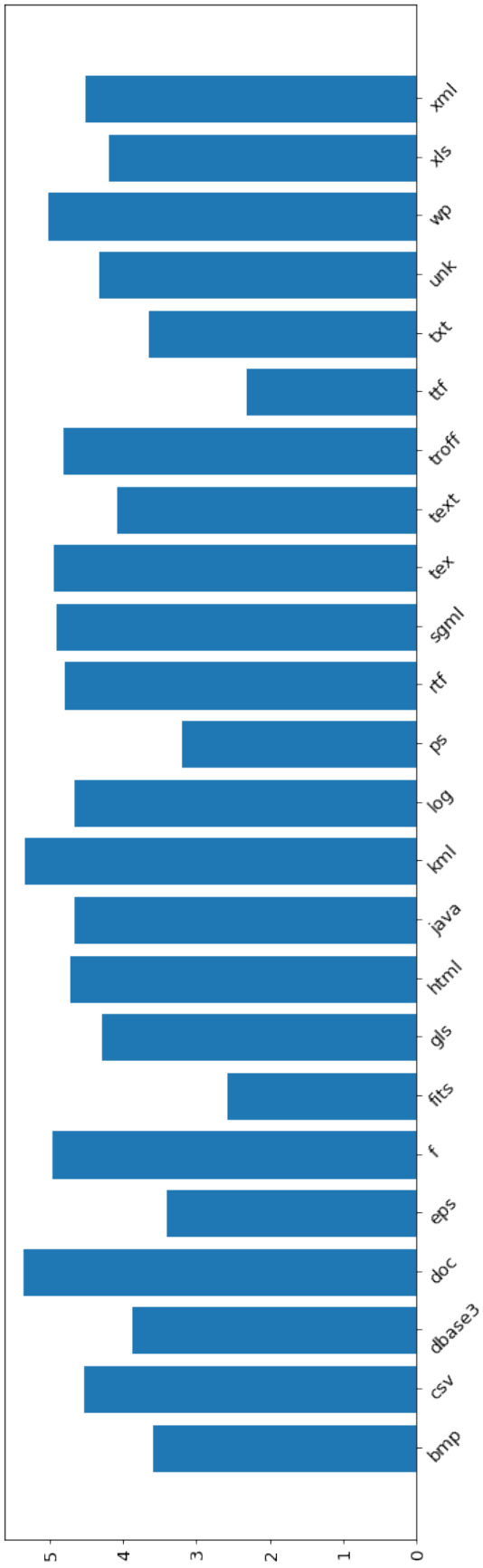

Figure 6.8: Average entropy of file types below 5.5 - benign traffic

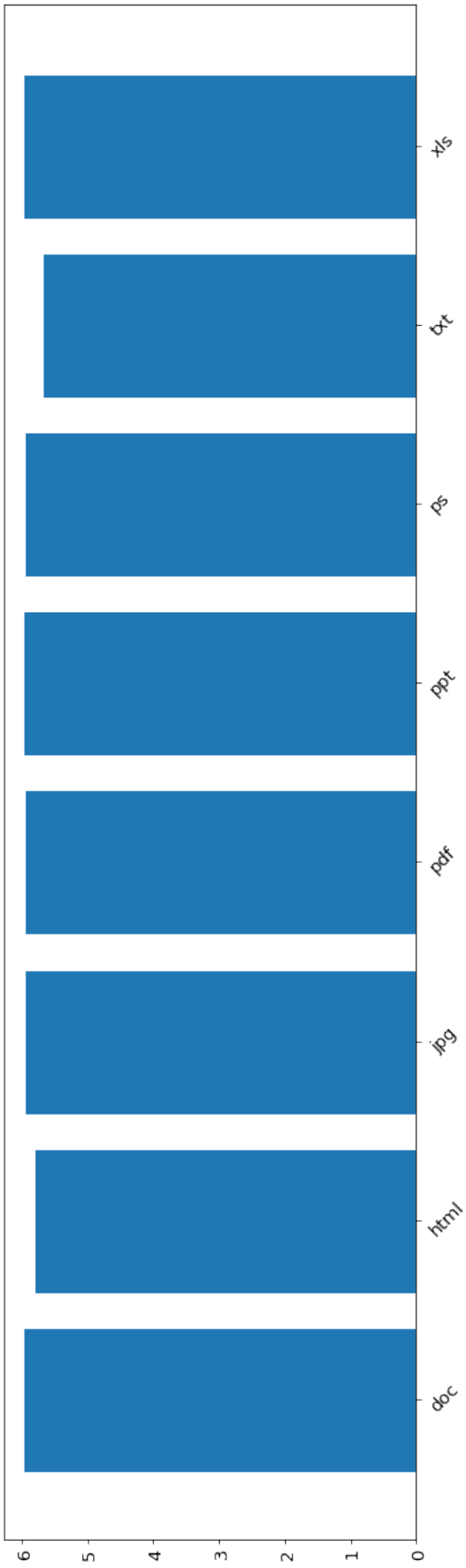

Figure 6.9: Average entropy of file types above 5.5 - benign traffic

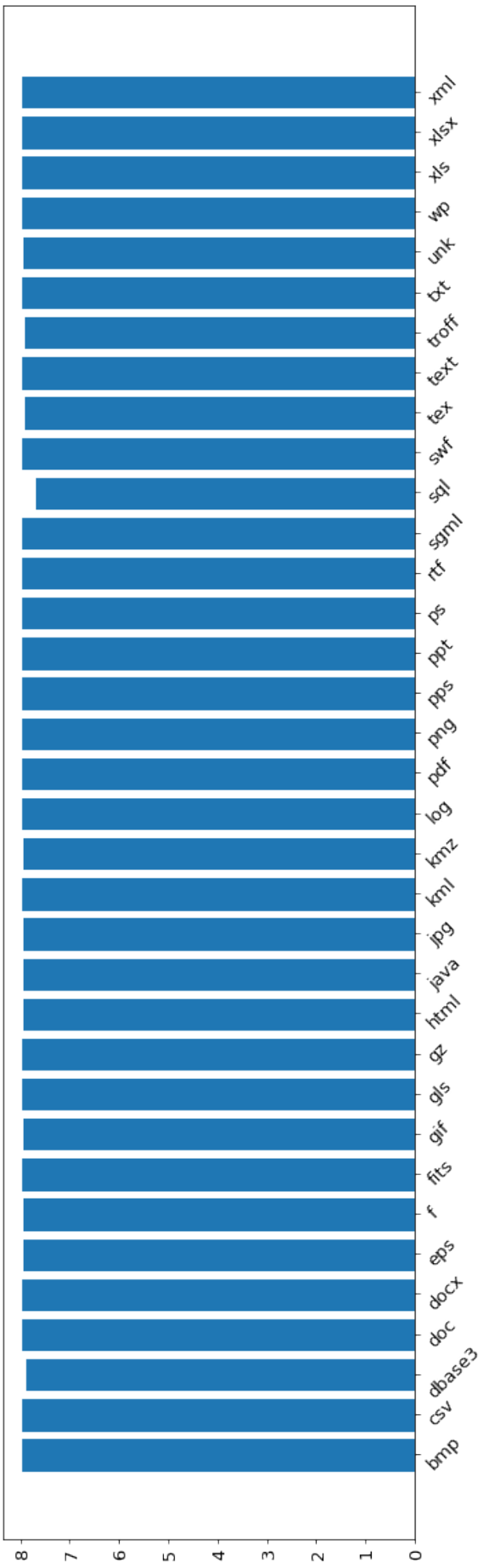

Figure 6.10: DarkRadiation average write entropy for all file types

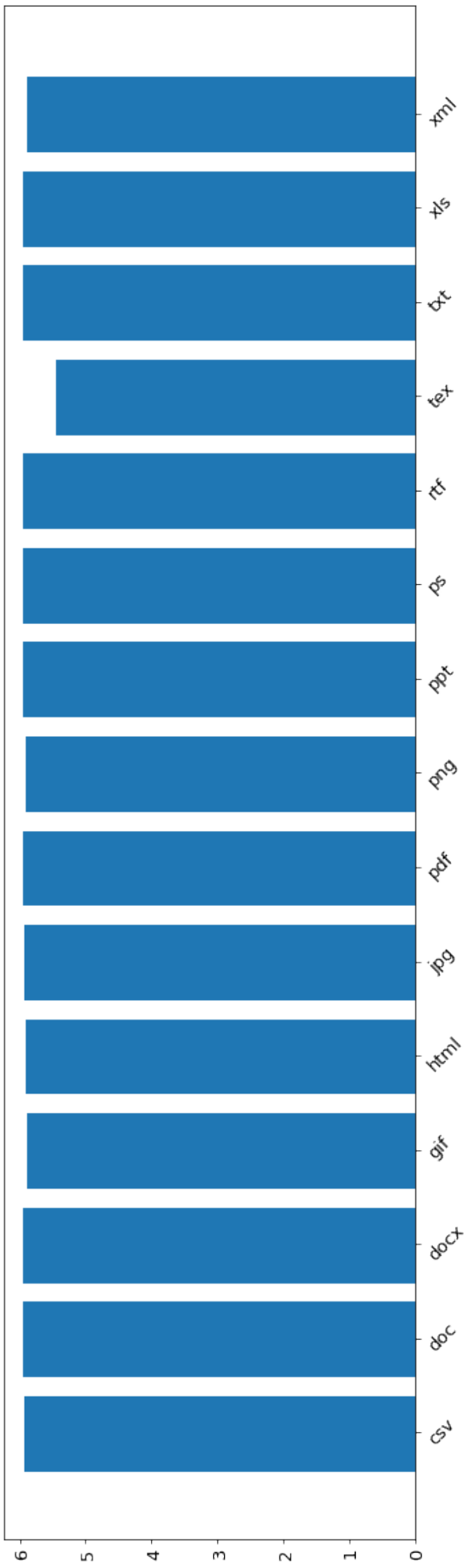

Figure 6.11: Ransomware PoC average write entropy for all file types

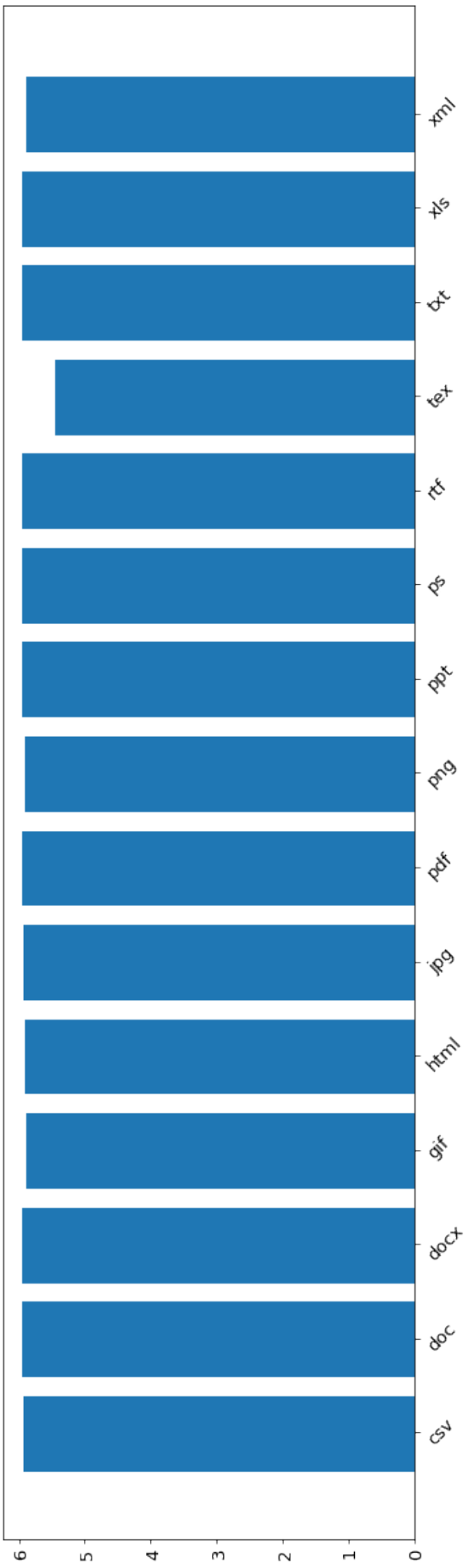

Figure 6.12: Roar average write entropy for all file types

6.2 Classification Accuracy

Firstly, the accuracy has been evaluated by leaving out one dataset of a specific Ransomware. This how the model performs on samples that it has not seen and has not been trained before.This exercise was done on each individual Ransomware, so that in the end three models were trained and were have evaluated separately for each time period. Taking into consideration 3 time windows, in the end there were nine models trained in total. The models trained in this way were logistic regression and random forest classifier.

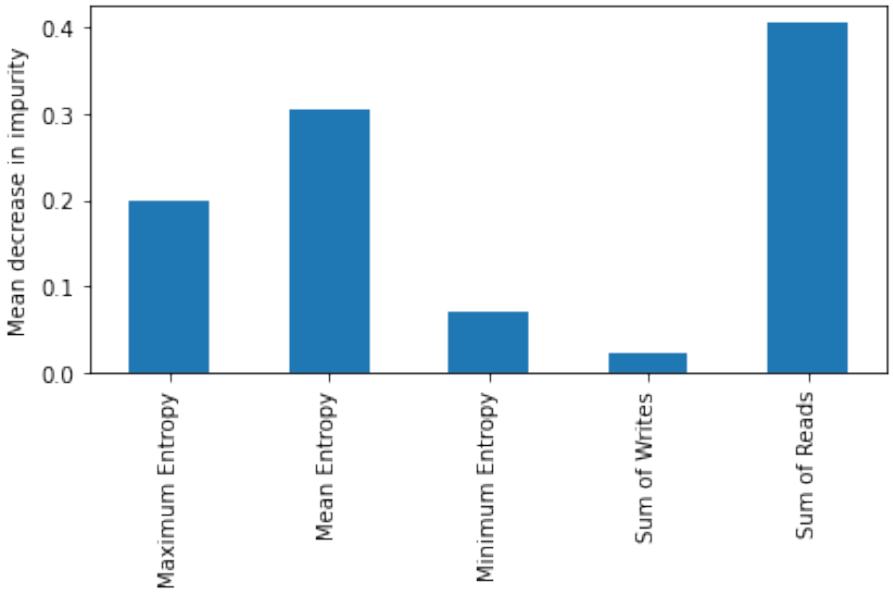

Figure 6.13: Feature Importance of Random Forest Classifier

As can be seen from Table 6.4, the accuracy of random forest classifier was the highest in time windows 5 and 10 seconds. At 10-second time window, the accuracy for Ransomware PoC and Roar has reached 100% and DarkRadiation 96.92%. As expected, the accuracy was the lowest on 2-second time window, albeit with only a marginal difference compared to other two time windows. This may be explained by the fact that on 2 seconds aggregation level, there were not enough file system operations to correctly predict whether there is malicious activity taking place. 5-second time window has shown slightly higher accuracy compared to 2-second time window. Although in 5-second time period, the detection of Roar is lower compared to 10-second time window, it is likely that evaluating the 5-second time window 2 times over 10 seconds would provide accuracy close to the 10-second time window model. The lower accuracy in the case of DarkRadiation compared to other two Ransomware samples is possibly because DarkRadiation is running multiple processes, whereas Ransomware PoC and Roar only one. During aggregation, grouping by PID and time window will result in one record per time window for Ransomware PoC and Roar, whereas for DarkRadiation multiple records will be present for each time window. This may result in different patterns present in the aggregated dataset, and the classifier might have difficulties correctly classifying DarkRadiation malicious samples. Another possible explanation is that the average entropy of Ransomware PoC and Roar is 6 (Figure 6.11 and Figure 6.12), whereas in the case of DarkRadiation it is around 8

(Figure 6.12). However, random forest classifier is relatively robust to these differences, as the accuracy is above 94% for all time windows considered. Additionally, the trained random forest classifier has shown accuracy of 100.0% across all time windows and across all left-out Ransomware samples (Table 6.1).

Table 6.1: Accuracy of Benign Workload Detection - Random Forest Classifier

	Left-Out Ransomware		
Time Window	PoC	DarkRadiation	Roar
2s	100.0%	100.00%	100.0%
5s	100.0%	100.0%	100.0%
10s	100.0%	100.0%	100.0%

In the case of logistic regression (Table 6.2), Ransomware PoC has been correctly classified in 100% of the cases. On the other hand, for DarkRadiation and Roar it has consistently underperformed compared to random forest classifier. For DarkRadiation moving only at around 38% accuracy and for Roar ranging from 41% up to 85% accuracy. Similarly to random forest classifier, the classifier has shown the lowest performance on DarkRadiation. The reasoning is in essence also the same, where the average entropy is much higher in case of DarkRadiation compared to Ransomware PoC and Roar. However, in the case of logistic regression, the degradation in performance is much more prevalent compared to random forest classifier. Unseen benign workload has been consistently around 100% for all time windows and all left-out Ransomware datasets (Figure 6.1)

Table 6.2: Accuracy for unseen Ransomware - Logistic Regression

Time Window	*Ransomware PoC*	*DarkRadiation*	*Roar*
2s	100.0%	38.85%	85.09%
5s	100.0%	44.31%	41.49%
10s	100.0%	33.61%	56.41%

Training the models for anomaly detection using isolation forest, all three Ransomware samples are unseen, as the model has been trained only on non-anomalous (benign) dataset. For All time windows, the results are very similar, moving around 100% for both Ransomware PoC and DarkRadiation, whereas reaching approximately 83% on Roar sample across all 3 time windows. The benign workload has been correctly predicted as benign around 80% of the time (Table 6.3). This is suboptimal, since if the model should be deployed on a real machine, low accuracy for benign workload would raise many false positives and the MTD would be falsely initiated frequently. Similarly to random forest classifier, logistic regression has also shown high accuracy in correctly classifying benign workload across all time windows.

The last approach was to concatenate all Ransomware and benign datasets together and split them into 80% train and 20% test set. Both logistic regression and random forest classifier have shown high accuracy - close to 100% (Figure 6.6). In Figure 6.13 the feature importance is depicted for random forest classifier. Sum of reads is the most important

feature for random forest classifier, followed by mean entropy. Minimum entropy and sum of writes are not that significant, likely because the feature sum of writes is correlated with the sum of reads and minimum entropy with the rest 2 Entropy measures.

6.3 Speed of Detection

Even though the prediction accuracy is high, it is also important to consider the speed at which the detection system can classify whether malicious activity is present or not. This is especially crucial, since the more time is needed for detection, the higher the number of irreversibly encrypted files.

The random forest classifier model trained on 5 seconds time window has been deployed on Raspberry Pi. The time for the end-to-end detection system to detect the Ransomware and initiate MTD has been measured. The speed of detection was measured on all 3 Ransomware samples. In total, for each sample, the Ransomware has been run five times and measured the time it took to detect. The minimum, average and maximum detection time have been measured. Similar amount of time has elapsed for Ransomware PoC and DarkRadiation to be detected, where minimum was 4 seconds and maximum 10 seconds for both samples. The longest time needed for detection was in case of Roar, where the average time is 19 seconds, minimum 8 seconds and maximum 44 seconds. This can be explained by the fact that Roar has a much slower encryption rate than the other two Ransomware samples (Table 6.7).

Although it may take a few seconds to detect the Ransomware, another interesting metric is how many files have been encrypted during until the Ransomware has been finally detected. Although Roar on average needed the longest time for detection, it has encrypted the least amount of files, namely 3 files. In comparison, DarkRadiation encrypted 35 files before detection and Ransomware PoC 18 files (Table 6.8). This is consistent with the speed of writes of each Ransomware, as can be seen in Figure 6.6. DarkRadiation has managed to encrypt the highest amount of files and at the same time has the highest encryption rate. On the other hand, Roar has very low encryption rate and has encrypted only 3 files in the timespan where the Ransomware ran undetected.

Table 6.3: Anomaly Detection Accuracy - Isolation Forest

Time Window	*Ransomware PoC*	*DarkRadiation*	*Roar*	*Benign Workload*
2s	99.9%	99.98%	82.8%	76.2%
5s	100.0%	100.0%	83.57%	80.61%
10s	100.0%	100.0%	83.71%	81.57%

6.4 Performance Considerations

Benchmarks were made to compare how many resources different states of the system take with both with and without the detection system running. In the two result tables (Table 6.9 and 6.10) different metrics can be observed, namely CPU time spent in user space, CPU time spent in kernel space as well as RAM usage. There are 4 different states of the Raspberry Pi machine. The machine can either be in idle state, FTP traffic present on the machine, Ransomware running on the machine and finally FTP traffic running concurrently with Ransomware.

In the idle state, it is expected that both with and without running the detection system, the usage is similar, as there are no entropy calculations taking place and no model predictions taking place if there is no file system activity present. From the tables there is no obvious difference visible for all metrics and all states.

However, resource usage itself may not directly show how much overhead is present by having the detection system running in the background. The detection system might take up resources at the cost of lower speed of reads or writes. To test this, three zip files have been selected, having size in total 1.1GB and copied the zip files into the mounted directory with detection system running. Secondly, the same files have been uploaded to arbitrary directory without the detection system. Writing of zip files took without MTD 4 minutes and 4 seconds, whereas the same files took 8 minutes and 40 seconds, thus increasing the time needed for writing two-fold.

Table 6.4: Accuracy for unseen Ransomware - Random Forest Classifier

Time Window	*Ransomware PoC*	*DarkRadiation*	*Roar*
2s	100.0%	94.71%	99.43%
5s	100.0%	95.55%	99.65%
10s	100.0%	96.92%	100%

Table 6.5: Accuracy of Benign Workload Detection - Logistic Regression

Left-Out Ransomware			
Time Window	Ransomware PoC	DarkRadiation	Roar
2s	99.95%	100.00%	99.95%
5s	99.98%	100.0%	99.98%
10s	99.98%	100.0%	99.98%

Table 6.6: Train Test Split Accuracy

Algorithm	*2s*	*5s*	*10s*
Logistic Regression	99.53%	99.76%	99.87%
Random Forest Classifier	99.93%	99.97%	99.98%

Table 6.7: Average Time Elapsed Before Detection (5s model)

Ransomware Sample	Minimum	Average	Maximum
Ransomware PoC	4 seconds	6 seconds	10 seconds
DarkRadiation	4 seconds	8 seconds	10 seconds
Roar	8 seconds	19 seconds	44 seconds

Table 6.8: Number of Files Encrypted Before Detection

Ransomware PoC	DarkRadiation	Roar
18 files	35 files	3 files

Table 6.9: Resource Usage Without Detection System

Resource Type	Idle	FTP Traffic only	Ransomware Only	FTP Traffic with Ransomware
CPU user space	8%-10%	21%-42%	33%-36%	57%-67%
CPU kernel space	14%-17%	28%-34%	16%-20%	30%-32%
RAM used	253MB	314MB	284MB	311MB
RAM free	850MB	531MB	824MB	705MB
RAM buffer/cache	741MB	1022MB	722MB	798MB

Table 6.10: Resource Usage With Detection System

Resource Type	Idle	FTP Traffic only	Ransomware Only	FTP Traffic with Ransomware
CPU user space	7%-10%	32%-48%	32%-34%	56%-60%
CPU kernel space	13%-17%	27%-32%	18%-22%	30%-32%
RAM used	303MB	375MB	284MB	320MB
RAM free	740MB	516MB	824MB	877MB
RAM buffer/cache	945MB	1022MB	860MB	662MB

Chapter 7

Summary and Conclusions

The content of this thesis is summarized in this chapter. The limitations of the implemented system are discussed and possible directions of future work are proposed.

7.1 Limitations

In case of file formats that have high entropy on average (above 5.5), the model ignores that write operation. In case of a Ransomware that focuses only on high entropy file types, such as images, videos, compressed files, the detection system would no longer be effective. In this case, the model will never make any predictions, since those records are filtered and the MTD will not get initiated. Additionally, although the detection and mitigation system does not show any significant increase in CPU and memory usage, it significantly slows down the speed of writing into the overlay file system. The time needed to write into the file system was doubled. Although the Ransomware activity is detected within few seconds, certain amount of user data loss is still present, as the model evaluates the activity at specific time intervals. Finally, the overlay file system has been running in a specific FTP folder. However, Ransomware usually encrypts the user's home directory. In such case, all files except the specific directory in which the overlay system is running would be encrypted by the Ransomware.

7.2 Conclusions

In this thesis, a real-time Ransomware detection and mitigation system has been proposed, designed and evaluated. Overlay file system has been designed to log all raw file system operations into a CSV file. The behavior of three Ransomware samples have been analyzed with regard to entropy of file writes, as well as the amount of reads and writes. The Ransomware behavior was put in contrast with benign workload, which was in this case FTP traffic. JMeter has been utilized to simulate benign file system activity. Varying rate of FTP uploads and downloads was used to mimic real-world settings of an FTP server.

Additionally, the same overlay file system has been used to mitigate a Ransomware attack by changing the file name after the file has been read by a process, rendering it impossible for the Ransomware to successfully perform encryption of user data.

Three different ML algorithms have been used to dynamically detect Ransomware based on the file system operations. All three algorithms have shown high accuracy, with random forest classifier having superior detection rate, where both Ransomware behavior as well as benign behavior classification has shown accuracy close to 100%.

The random forest classifier model trained on 5 seconds aggregation period has been deployed on the Raspberry Pi device. The final detection and mitigation system interact together to prevent the encryption of user data. Overlay file system is logging all file system operations, which are then fed into the model that evaluates whether the file system activity is benign or malicious. After detection, the overlay file system is informed and initiates the MTD strategy.

Finally, the speed of detection and performance overhead of the deployed detection system has been monitored and evaluated. The benchmark was the system running without an overlay file system and detection system. No significant differences between the detection/mitigation system and the benchmark have been detected with regard to CPU and memory usage. However, the speed of write operations has been significantly decreased with the detection and mitigation system running. Although the detection system is highly accurate, the Ransomware samples have encrypted from 3 up to 35 files before the system detected any malicious activity.

7.3 Future Work

Future work could further improve the detection system by making the system robust against Ransomware encrypting specifically high entropy file formats. For instance, instead of filtering out high entropy file extensions, a dummy entropy value could be set. In that case, the model would still evaluate the batch of records, however the model would rely only on the number of write and read operations. Inventing new features for the model could also increase the robustness of the model. For instance, similarity measure of the file before the write operation and the modified file after the write operation. Another aspect to consider is the data buffering strategy. Assuming the model is robust against the encryption of high entropy file formats, implementation of buffering as proposed in 4, the system could in theory prevent any file encryption altogether. Finally, adjusting the overlay file system to initiate the root directory in user's home directory would protect all files and directories of the user.

Bibliography

[1] *Internet of things (iot) connected devices installed base worldwide from 2015 to 2025*, `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`, Accessed: 2023-04-01.

[2] A. Ifran, M. Niazy, R. Ziar, and K. S., "Survey on iot: Security threats and applications", *Journal of Robotics and Control (JRC)*, vol. 2, no. 1, pp. 1–2, 2021.

[3] M. F., C. M. Z. D., P. M., and Z. A., "Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices", *INTERNET OF THINGS JOURNAL*, vol. 6, no. 5, pp. 8182–8201, 2019.

[4] A. M., A. T. B. M., B. M., *et al.*, "Understanding the mirai botnet", *USENIX security symposium*, vol. 26, no. 17, pp. 1093–1110, 2017.

[5] G. A. Conti M. and R. S., "On the economic significance of ransomware campaigns: A bitcoin transactions perspective", *Computers Security*, vol. 79, pp. 162–189, 2018.

[6] L. K., L. S.Y., and Y. K., "Machine learning based file entropy analysis for ransomware detection in backup systems", *IEEE Access*, vol. 7, pp. 110 205–110 215, 2019.

[7] O. G. and G. McDonald, "Ransomware: A growing menace.", *Arizona, AZ, USA: Symantec Corporation*, 2012.

[8] D. Huang, V. Aliapoulios M.M.and Li, L. Invernizzi, *et al.*, "Tracking ransomware end-to-end", *IEEE*, pp. 618–631, 2018.

[9] G. S., K. S., H. K., G. Martin, D. A., and A. P., "A retrospective impact analysis of the wannacry cyberattack on the nhs", *NPJ digital medicine*, vol. 2, p. 98, 1 2019.

[10] A. M. and R. A., "A note on different types of ransomware attacks", *Cryptology ePrint Archive*, 2019.

[11] S. Alty, S. Millasseau, P. Chowienczyk, and J. A., "Cardiovascular disease prediction using support vector machines", *Midwest Symposium on Circuits and Systems*, vol. 1, pp. 376–379, 2003.

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

[13] K. S.B., "Decision tree methods: Applications for classification and prediction", *Shanghai archives of psychiatry*, vol. 27, pp. 130–135, 2 2015.

[14] Z. L., B. T.R., and C. R., "Lessons from infant learning for unsupervised machine learning", *Nature Machine Intelligence*, vol. 4, pp. 510–520, 6 2022.

[15] C. V., B. A., and K. V., "Anomaly detection: A survey", *ACM computing surveys*, vol. 41, pp. 1–58, 3 2009.

[16] Z. R., D. S.A., and O. X., "Towards a theory of moving target defense", *Proceedings of the first ACM workshop on moving target defense*, pp. 31–40, 2014.

[17] C. G., W. B., and W. T., "Moving target defense: State of the art and characteristics", *Frontiers of Information Technology Electronic Engineering*, vol. 17, pp. 1122–1153, 11 2016.

[18] J. J.D., A.-S. E., and D. Q., "Openflow random host mutation: Transparent moving target defense using software defined networking", *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 127–132, 2012.

[19] J. Q., S. K., and S. A., "Motag: Moving target defense against internet denial of service attacks", *22nd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, 2013.

[20] V. B.K.R., T. V., and Z. E., "To fuse or not to fuse: Performance of user-space file systems", *FAST*, vol. 17, pp. 59–72, 2017.

[21] S. N., C. H., T. P., and B. K.R.B., "Cryptolock (and drop it): Stopping ransomware attacks on user data", *IEEE 36th international conference on distributed computing systems*, pp. 303–312, 2016.

[22] R. V., "Data fingerprinting with similarity digests", *Advances in Digital Forensics VI: Sixth IFIP WG*, pp. 207–226, 2010.

[23] K. A. and K. E., "Redemption: Real-time protection against ransomware at end-hosts", *Research in Attacks, Intrusions, and Defenses: 20th International Symposium*, pp. 98–119, 2017.

[24] T. F., M. B., L. J., Z. F., S. J., and M. J., "Ransomspector: An introspection-based approach to detect crypto ransomware", *Computers Security*, vol. 97, p. 101 997, 2020.

[25] A. Continella, A. Guagnelli, G. Zingaro G. amd De Pasquale, A. Barenghi, Z. S., and F. Maggi, "Shieldfs: A self-healing, ransomware-aware filesystem for ransomware detection in backup systems", *Proceedings of the 32nd annual conference on computer security applications*, vol. 7, pp. 336–347, 2016.

[26] S. R., "Design and implementation of a virtual file system for hostbased moving target defence in iot devices", 2022.

[27] *Digital corpora data corpus*, `http://digitalcorpora.org/`, Accessed: 2023-04-14.

[28] *Go fuse library*, `https://github.com/hanwen/go-fuse/`, Accessed: 2023-04-14.

[29] L. J., "Ai-powered ransomware to optimize its impact on iot spectrum sensors.", 2023.

[30] *Jmeter*, `https://jmeter.apache.org/`, Accessed: 2023-04-14.

[31] B. L., "Random forests", *Machine Learning*, pp. 5–32, 2001.

[32] H. D.W. and L. S., *Applied Logistic Regression*. John Wiley Sons, 2013.

[33] L. F.T., T. K.M., and Z. Z., "Isolation forest", *eighth ieee international conference on data mining*, pp. 413–422, 2008.

Abbreviations

CSV	comma-separated values
FUSE	filesystem in userspace
FTP	file transfer protocol
IoT	Internet of Things
IRP	I/O request packet
ML	machine learning
MTD	Moving Target Defense
PID	process identifier
RAM	random-access memory

List of Figures

List of Tables

List of Algorithms

Appendix A

Files Collected from Data Corpus

This chapter references all files from Digital Corpora [27] that have been used for collection of Ransomware as well as benign behavior.

- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/000.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/001.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/002.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/003.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/004.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/005.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/006.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/007.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/008.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/009.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/010.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/011.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/012.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/013.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/014.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/015.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/016.zip

- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/017.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/018.zip
- https://digitalcorpora.s3.amazonaws.com/corpora/files/govdocs1/zipfiles/019.zip

Appendix B

Packages Used

All the GitHub links to the relevant packages that have been used throughout the thesis are listed below.

- https://github.com/hanwen/go-fuse
- https://github.com/scikit-learn/scikit-learn
- https://github.com/matplotlib/matplotlib
- https://github.com/pandas-dev/pandas
- https://github.com/numpy/numpy