University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Cybersecurity

Ransomware Detection Techniques: A Comparative Study

Supervisor
Prof. Mauro Conti
University of Padova

Co-supervisor
Prof. Fabio De Gaspari
Sapienza University of Rome

Master Candidate
Giulio Umbrella

Student ID
2021132

Academic Year
2023-2024

"TANSTAAFL: There ain't no such thing as a free lunch"
— Robert A. Heinlein: The Moon Is a Harsh Mistress

Abstract

The increasing threat posed by ransomware has emphasized the necessity of automated tools for identifying such malware in a system. The current state-of-the-art in ransomware detection relies heavily on monitoring file operations to identify malicious encryption processes. Several models have been developed to detect ransomware effectively. However, as new types of ransomware are continually developed, existing models may experience deteriorating performance. This phenomenon, known as dataset shift, causes the accuracy of these models to decline over time. To study the effects of dataset shift on ransomware detection, it is crucial to create a dataset that encompasses all the necessary features. In the context of my thesis, I propose a framework for automating the creation of a dataset from ransomware samples. This framework begins with a list of ransomware samples, which are detonated in a secure environment to extract relevant data. This work lays the foundation for future research aimed at testing the robustness of detection models over time.

Contents

Listing of figures

Listing of tables

Listing of acronyms

API . . . . . Application Programming Interface

IDS . . . . . Intrusion Detection System

IRP . . . . . I/O request packets

ML . . . . . Machine Learning

OS . . . . . Operating System

SotA . . . . . State of the Art

1 Introduction

In recent years, ransomware has become a pervasive threat in the cybersecurity landscape. Ransomware is a type of malware that encrypts files on a victim's system, rendering them inaccessible until a ransom is paid for the decryption keys. This form of malware has not only evolved in terms of tactics but also as a business model, with the rise of the Ransomware-as-a-service model where criminal groups develop new ransomware variants and lease them to other groups to carry out attacks[1]. Ransomware is responsible for several incidents among many different institutions, for a total loss estimated in the billions [2]. In 2022, 68 percent of the worldwide reported cyberattacks were ransomware and in the fourth quarter of 2022, nearly 155 million ransomware attacks were detected worldwide [3].

Typically, attackers trick users into downloading a malicious application onto their system, which then executes a payload to encrypt files and subsequently blackmails the user by threatening to permanently deny access to the files [4]. In this thesis, we focus on the actions performed by ransomware during the malicious encryption of the files. One of the main issues with ransomware is that encryption itself is not inherently a malicious action, as it is routinely performed by legitimate processes within a system. The malicious nature of ransomware arises from its indiscriminate use of encryption to prevent users from accessing their files [5]. To identify ransomware, the literature has proposed two main approaches: signature-based analysis and behavioral-based analysis [4]. Signature-based detection performs a static analysis of ransomware by comparing it to a known database of malware signatures; while this technique is fast, it cannot identify new or unknown threats. In contrast, behavioral-based detection

builds a model to dynamically monitor the actions of ransomware and compares these actions against a baseline model. Although this technique is more prone to false positives, it can better detect the presence of new or evolving attacks. Due to the intrinsic limitations of signature-based approaches, behavioral-based methods are the most common solution for identifying ransomware[6].

Researchers have proposed several Intrusion Detection Systems (IDS) to detect the presence of ransomware in a system [7]. An IDS is a software system that automates the detection of intrusion in a system, where an intrusion is a malicious attempt to tamper with the confidentiality, integrity, or availability (CIA) [8] of the system. A behavioral IDS monitors the system's action as it interacts with the system and uses a model to classify its behavior as legitimate or malicious. As mentioned earlier, the most popular solution for detecting ransomware in a system is behavior-based detection. An IDS monitors how processes interact with the system to identify malicious behavior. The IDS can use three main types of dynamic features to monitor the processes: API calls and file operations [2]. In this work, we focused on the IDSs based on the file operations [6] [5] [9] [10] [11] as they are considered the State of the Art (SotA) IDS. These IDSs focus on how files are modified within the system to identify malicious encryption patterns. Thus, the emphasis is on the operations performed on the files, rather than the actions taken by the process itself.

Usually, the models are derived from historical observations of known benign and malicious entities. However, the performance of the models may deteriorate over time due to the dataset shift problem [12]. This problem occurs when the reality the dataset describes evolves over time, while the model remains the same. Hence, the model cannot capture the new dynamics, and may not perform as well as before. The SotA classifiers in ransomware detection do not account for this issue, and as a result, their claimed performance may degrade over time. To address this issue, this work proposes a framework for collecting relevant data to reproduce SotA models and use them as a basis for investigating the effect of dataset shift on IDS performance. We first identify the SotA IDS for ransomware detection and list all the features used to develop these models. Then, we build a mechanism for automatically collecting the required data from ransomware samples. This mechanism takes a list of ransomware samples as input, detonates each of them in a sandbox environment, and collects all the necessary data. We evaluate the quality of the framework by identifying ransomware samples suitable for analysis.

1.1 Organization

Chapter 2 provides background information on ransomware and the main phases of an attack. We then introduce the concept of IDS and the different types of detection methods available. Next, we review the SotA IDS approaches, explaining how they propose to detect ransomware and the types of features they use. We then review all the required features to train the models. We also discuss the "data shift" problem. Chapter 3 describes the framework proposed in this work, offering an overview of its mechanism and three main components, along with details on the automated data collection process. We also explain how we obtained the ransomware samples and utilized the proposed framework to identify which samples successfully detonated. Chapter 4 provides additional details on the sandbox component and the input/output operations in Windows. Finally, Chapter 5 concludes the thesis by summarizing the main contributions and lessons learned and suggesting future research topics.

2
Background

In this chapter, we present the background and related work of this thesis. Section 2.1 illustrates what ransomware is and how it operates. Next, Section 2.2 explains the Intrusion Detection System (IDS), its role in cybersecurity, and its two main approaches, Signature and Dynamical. Section 2.3 provides more details about the SoA IDS for ransomware, explaining the models and features they use to identify ransomware. Section 2.4 reviews the features required for training these models. Finally, Section 2.5 introduces the dataset shift problem and explains its significance for IDS performance.

2.1 Ransomware

Ransomware is a type of malware that uses cryptography to block users from accessing their data, demanding a ransom payment to restore access to the system [13]. Ransomware has grown as a preeminent threat in cybersecurity. In 2017, the Wannacry ransomware [14] spread to more than 200,000 computers in over 150 countries. Ransomware activities have a significant impact on businesses with estimated payments amounting to hundreds of millions of US Dollars [15]. Besides the financial cost, ransomware has caused delays in critical infrastructure services; in 2023, several hospitals in Veneto have been targeted by ransomware, delaying their activities [16].

Ransomware is generally categorized into two groups: Locker ransomware and Crypto ransomware [17]. Locker ransomware denies access to the workstation's capability, restricting

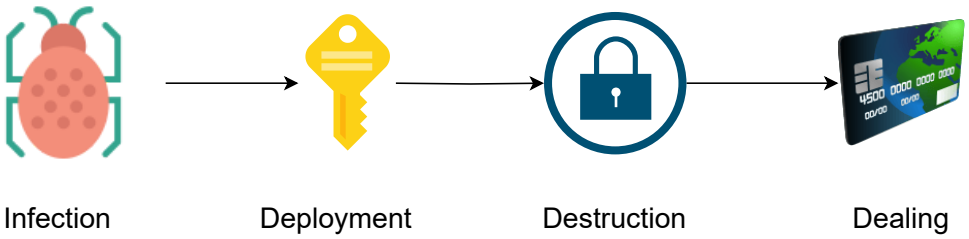


Figure 2.1: General attack scheme for ransomware: In the infection phase, the machine is compromised; in the deployment phase, ransomware generates the secret keys; in the destruction phase, it encrypts the files; in the dealing phase, the ransomware alters the user and provides the payment instruction.

user's actions to the bare minimum required to pay the ransom. The *Trojan.Randsom.C* ransomware [17] displayed a fake Windows security center screen, falsely claiming that the user's license has expired and providing instructions for the payment. However, as the underlying files are not compromised, the ransomware could be removed and access to the resource restored, making this technique less effective for extorting money. Due to this deficiency in the extortion scheme, 2013 [17] saw a rise in the diffusion of Crypto ransomware. Crypto ransomware encrypts user files by scanning the system and making their contents inaccessible without the decryption key. In 2022 [18], all the top ransomware attacks were of the Crypto type. Due to their prominent role, we focus on crypto-ransomware in this work and refer to them as ransomware.

Figure 2.1 shows a general scheme of Crypto ransomware actions [4] which can be divided into four steps [4]: delivery, deployment, destruction, and dealing. In the delivery phase, the ransomware attempts to reach the victim's system. In the deployment phase, the ransomware gathers all the required resources for destruction; In the destruction phase, the ransomware executes its payload. In the dealing, the ransomware displays a ransom note demanding payment to decrypt the files.

The **delivery** phase can occur through several methods [17]: malicious advertisement, span email, Downloader and Botnet, social engineering, and self-propagation. With malicious advertisements, the attacker creates malicious web pages linked to URLs that distribute ransomware. Additionally, the attacker can pay a Traffic Distribution Service to redirect web traffic to these malicious addresses; in this scenario, the user clicks on a legitimate service, but the web server redirects it to the malicious page instead. Span email employs social engineering tactics, such as sending false mail delivery notifications, to trick the user into downloading ransomware or visiting a malicious web page to download it. The Downloader is a more sophisticated tactic

that requires collaboration among different groups; the first group initiates a campaign to install downloader softwares on the victim's machine. The download remains inactive initially, waiting for instructions. The collection of compromise machines is called a botnet. The group rents the botnet infrastructure to other groups, which can use the downloaders to distribute the ransomware. The attacker can incorporate a self-propagation mechanism in ransomware, allowing it to automatically spread across networks. However, this strategy could result in multiple infections of the same machine, potentially discouraging the victim from paying the ransom. Another form of collaboration between criminal groups is the Ransomware-as-a-service, where a group develops ransomware and infrastructure for operations and then rents it to other groups that manage the diffusion of the ransomware.

In the **deployment** phase, the ransomware prepares for the next phase by gathering all the necessary resources required to execute the attack. As an example, Wannacry [19] [20] establish a persistence mechanism to ensure that the Operating System (OS) executes it at start-up. Wannacry also includes a kill switch, a mechanism to exit without performing the destruction phase[20]. The ransomware makes a web request to a hard-coded domain; if it receives a response, it blocks execution.

During the **destruction** phase, the ransomware encrypts the victim's files. Attackers have various options for both the steps for substituting files and the encryption scheme used. In [6], authors categorize ransomware into three classes based on the mechanism for substituting the original file. The basic steps for replacing a file typically involve reading the content of a file, encrypting it, writing it in a file, and removing the original file. Based on these steps, the authors identify three categories of ransomware: **Class A** ransomware encrypts the file's content in place by reading the file, encrypting it, and overwriting the original file with the encrypted content. **Class B** is similar to class A, but with an additional step: it moves the original file to a temporary directory before overwriting it with the encrypted contents and then it sends it back to the original position. **Class C** reads the file's contents and then creates a new file where to save the encrypted content; it then deletes the original file or replaces it with the encrypted. Another way to classify ransomware is by using the encryption mechanism to encrypt files.

Another important consideration in ransomware design is the encryption scheme used to encrypt the victim's files [17] [21]. Ransomware may employ symmetric encryption, asymmetric encryption, or a combination of the two. Symmetric encryption uses a single key for both encryption and decryption; symmetric is fast but requires careful key management to prevent the victim from recovering the files. If the ransomware stores the key locally on the

victim's machine, there is the risk that the victim could find it and decrypt the files. Alternatively, ransomware may generate a key on the infected machine and send it to the attacker, but this requires the infected computer to be connected to the network. Asymmetric encryption involves two separate keys: a public key for decryption and a private key for encryption. The attacker can embed the public key within ransomware without sharing the private key. However, a victim may obtain and spread the private key to other victims. To mitigate this, the ransomware may generate a private-public key pair for each computer, sending the private key to the attacker's server. However, asymmetric encryption is generally slower than symmetric, increasing the chances of detection by an intrusion detection system (IDS), and requiring the infected computer to be connected to a network. A third option is the combination of symmetric and asymmetric. The attacker generates a server-side pair of private-public keys and embeds the public key on the ransomware. When the ransomware infects a machine, it generates a client-side public-private key pair and a symmetric key on the victim's machine. The symmetric key is used to encrypt files and is then encrypted with the client's public key. The client's private key is subsequently encrypted with the server's public key. This process does not require a connection. When the victim pays the ransom, the attacker decrypts the client's private key with the server's private key, which decrypts the symmetric key, allowing the victim to recover their files.

In the **dealing** phase, the ransomware notifies the user about the encryption by altering the display image with a warning message [19] and providing instructions for the payment. Typically, the attacker demands payments in cryptocurrency, such as bitcoins, to make it difficult to trace the transaction. [13].

The classification of ransomware into families is a common approach in the literature, due to the complexity and variability of ransomware attacks. Each phase of a ransomware attack can be carried out in multiple ways, making organization and classification challenging. Ransomware families are the common code signatures, the viral payloads, and malicious commands that do the dirty work once they gain access to a business's systems [22]. So, a family constitutes the building block for creating ransomware, and criminal groups fill in the details according to their peculiar needs.

2.2 Intrusion Detection System

An Intrusion Detection System (IDS) [23] is a software or hardware system to automate the intrusion detection process. An intrusion is a malicious action that attempts to compromise

the Confidentiality, Integrity, and Availability (CIA) [8] or bypass the security of a computer or a network. Intrusion detection is monitoring events in a network or computer system to identify signs of intrusion. So, as reported in [7], the general goal of IDS is to protect a system that comes from network connectivity and the use of information systems. IDS employs two main approaches for the detection: signature-based and behavioral-based[7]. Both approaches have pros and cons [7] [23], and IDS tends to use a mixture of the two.

The **signature-based** detection method involves comparing the characteristics of an entity against a database of known malicious patterns or signatures. Detection is based on identifying matches with this database of known attacks. For example, in the case of ransomware, the hash of an executable file, such as its SHA-256 checksum, is compared against a database of known ransomware hashes. If a match is found, the executable is immediately identified as malicious. This approach is simple and highly effective in detecting known attacks, with a small number of false positives. However, signature-based attacks are limited to known attacks and perform poorly against unknown ones. Additionally, it requires continuous updates of the database of attack signatures, which can be time-consuming and difficult to manage.

The **behavioral-based** detection assumes that there are patterns of legitimate and abnormal behaviors in how an entity operates in a system. It relies on historical data to create a model that defies what is considered a legitimate behavior. The IDS then uses this model and uses his model to continuously monitor the actions of entities in the system. If an entity's behavior deviates from the established model of accepted behavior, the IDS flags it as potentially malicious.

In [7], authors describe three security-related scenarios where an IDS can be employed: scanning attacks and denial of service (DoS). In a **scanning** attack, the attacker tries to map the network's topology and service by sending various types of packets to hosts and ports. Although the attack does not cause harm to the system, it can expose sensitive knowledge about the network, which the attacker can use to identify vulnerabilities. An example of such an attack is TCP scanning, where the attacker sends several TCP packets with different port numbers to identify which services are listening on a host[24]. Another example is the OS detection function of the nmap software [25], which attempts to determine the OS running on a system by analyzing TCP and UDP responses. In both cases, the attacker sends an anomalous flow of packets to the target, which can be detected by the IDS. Snort, a popular open-source IDS, uses signature-based detection with filter rules to identify scanning activities[26]. Alternatively, in [27], the proposed IDS uses a neural network model based on behavioral data. A **DoS** attack aims to disrupt a target network or system by overwhelming it with a large volume of malicious

packets. The attacker can send a high number of packets, flood the target with a large amount of data, or send requests with spoofed IP addresses [28]. To mitigate such attacks, an IDS can monitor the rates of packet flow and block the traffic. In wireless sensor networks, authors proposed a behavioral rule-based IDS based on the consistency between transmission power and the sender's geographical position [29]

A behavioral IDS can use several techniques: rule-based, statistic-based, and machine learning [7] [30] [31].

Rule-based algorithms rely on prior knowledge of the expected entity's behavior. The rules can be derived from various sources, including the entity's underlying distribution, known behavioral models, or empirical observations. Rule-based anomaly detection is fast and simple and does not require any learning scheme or later updates to integrate new knowledge. However, rule-based models can be sensitive to noise in the observations, which may interfere with their ability to accurately compare behaviors against identified malicious patterns. Because of this, a rule is generally only effective in a specific application scenario. To improve detection accuracy, rule-based IDSs often employ a voting scheme that aggregates multiple rules, thereby enhancing their flexibility and robustness in different contexts.

Statistical algorithms are more flexible but require prior knowledge or have high complexity. They can be either parametric or nonparametric. Parametric techniques are easier to compute due to their underlying assumptions but can be biased by the choice of parameters. Nonparametric techniques, while more flexible and robust, require larger datasets.

Machine learning algorithms are generally complex, but do not require previous knowledge and are very effective. Due to the increasing popularity in cybersecurity [30] of ML-based algorithms, we discuss them in a dedicated section 2.2.1.

2.2.1 Machine learning model

Machine learning (ML) algorithms learn the legitimate profile by applying learning techniques on a dataset. Each row in the dataset represents a single observation about an entity and contains multiple pieces of information known as features. Features are characteristics that describe the entity and are typically numerical (e.g., a person's height) or categorical (e.g., nationality) [32]. The learning procedure can be supervised, unsupervised, or semisupervised [33]. In **supervised learning**, the dataset contains additional knowledge that is not available when the model is applied; for example, a dataset of emails categorized with the spam or non-spam labels [34]. This additional information reflects the contribution of an "expert in the field," as

the labels span and non-spam in the email dataset. The model is trained for a specific task, such as a classifier that assigns an observation to a category, or a regression model that predicts a continuous value. For instance, a spam filter is a classifier that predicts whether an email is spam or non-spam. The goal of supervised learning is to internalize the expert knowledge within the model and use it to make predictions about unseen instances. The dataset is split into in two components: a training set and a test set. The model is trained using the training set, usually by minimizing a cost function associated with the error made by the model. In the spam filter example, misclassifying a sample is considered an error, and the model is adjusted to minimize these errors. The model is computed to minimize the errors. However, this process may be affected by a bias called *overfitting*; if a model overfits a dataset, it becomes overly specialized in the observed data, capturing not only the patterns but also the noise or outliers. As a result, the model may not perform well when analyzing unseen observations. To mitigate overfitting, the training procedure includes regularization techniques that constrain the model and reduce its complexity[35]. The model performance is then evaluated against a test set containing labeled observations, unseen by the model during training. The model predicts an outcome that is compared against the known label. The quality of the model is assessed by its ability to correctly predict these labels. For the spam filter, this metric could be the number of correctly labeled emails. Unsupervised learning does not use any labels and works by summarizing the data to extract meaningful information, such as clustering the data in sub-groups with similar characteristics. Semi-supervised represents a middle ground, where the dataset contains some additional information, but not to the extent of fully labeled data as in supervised learning. ML-based algorithms for behavioral IDS have become increasingly popular in cybersecurity [30]. These models can handle large quantities of data [30] and generally achieve higher accuracy [36] [31]. However, ML has limitations [37]. Many ML models assume a balanced dataset, where each class has equal representation (e.g., the same number of spam and ham emails); an imbalanced dataset may lead to misleading results. As an example, assume that the dataset has 100 observations, 90 of class A and 10 of class B, and the task is classification. If the trained model always predicts class A, it will be right 90 % of the time. However, the model is not learning anything as it is not able to differentiate between the two classes. Translating this problem in an IDS for a computer, if the model predicts that all processes are legitimate, it will be right most of the time, but it will fail to detect a real threat. Another issue is the necessity for a labeled dataset for supervised models; creating such a dataset is expensive and time-consuming, which can limit the availability of relevant data. ML models also face security concerns; an attacker may poison the dataset by injecting malicious samples to impair the model's performance and

bias the model.

2.3 State of the art Ransomware IDS

As said above, ransomware is an increasing menace, so several IDSs have been proposed to identify them. In the context of IDS [23], as the ransomware goal is to encrypt the file, the primary threat is Availability. Some ransomware may also steal the data and menace the owner to publicly disclose or sell them, threatening Confidentiality [38].

To identify the State of the Art (SotA) works, we reviewed surveys on ransomware detection [2] [4] [39] [40]. We also reviewed some recently published papers to identify the most significant contributions [41] [42]. The identified works primarily focus on operations performed on files. Some models use an approach based on API calls, allowing them to track the complete actions of a process. However, this approach has much higher complexity because it monitors a wide range of activities. Focusing on files restricts the analysis to what ransomware actually does—altering the files. For these reasons, we selected the following papers as the most relevant and effective in dealing with ransomware.

2.3.1 Cryptolock

Cryptolock [6] is a behavioral rule-based IDS designed to monitor file activities. The authors identify three main indicators related to the malicious encryption of files and define a rule for each indicator. Each rule is either based on a specific *action* performed on a file or on a *metric* related to the modification of the file, and it is triggered under precise conditions that indicate a violation. Every time a process violates one of the rules, it commits an infraction; Cryptolock maintains a reputation score for each process and monitors its behavior during execution. If the process's score exceeds a threshold, Cryptolock flags it as malicious. Instead of tracking a process's actions, such as with API features, Cryptolock focuses on how the process manipulates files. While ransomware can hide its behavior, it is much harder to evade detection based on file manipulation checks. As with rule-based IDS, authors identified features linked to the encryption of files: file type change, similarity measurement, and Shannon's entropy.

To identify changes in the **file's type**, Cryptolocker monitors modification to its magic number. The magic number is a special byte sequence stored in the file's header that typically remains unchanged throughout the file's lifetime. If a process modifies the file extension, this is considered a violation.

Encryption transforms plain text into cipher text to ensure confidentiality; thus, plain and cipher texts should be very different. **Similarity** between two files measures the degree of resemblance between two files. Therefore, a plain cipher text pair should have a very low similarity. To compute similarity, authors use the sdhash function, which produces a score between 0 and 100, describing the confidence of similarity. Authors of the sdhash function provide guidelines to interpret the output [43]; a score of 0 suggests very low statistical significance, while a value of 100 indicates that the two files hare some common characteristics. A plain file and a ransomware-encrypted file should have a similarity close to zero, so a 0 value is a violation.

Entropy quantifies the average level of uncertainty with the variable [44]. For a random variable X which takes value over the set $\mathcal{X}$ the entropy is:

$$H(X) = \sum_{x \in \mathcal{X}} P(x_i) \log \frac{1}{P(x_i)} \tag{2.1}$$

Uniform probability distribution [45] has the highest amount of uncertainty and, therefore, has the highest associated entropy [46]. Entropy e of a file is calculated using the relative frequency of each byte:

$$e = \sum_{i=1}^{256} P_{B_i} \log_2 \frac{1}{P_{B_i}}, \text{where } P_{B_i} = \frac{\text{Frequency of byte i}}{\text{Total number of bytes}} \tag{2.2}$$

For a byte sequence, entropy assumes values between 0 and 8, where 0 represents a sequence composed of a single byte, and 8 is a sequence where every possible byte appears with the same frequency. Encryption and compression have high associated entropy, as these processes remove patterns and obscure the files' contents. Entropy is measured when a process writes on a file. During an atomic read-write of a file, Cryptolock computes a weight arithmetic mean of the entropies of the write P_{Write} and read P_{Read}; the weight w is defined as w = 1/8 * e * b, where e is the entropy, 1/8 a scaling factor, b is the total number of bytes. The weighted mean reduces the impact of low-entropy small read/write operations. The rule is considered violated when the difference in average entropies exceeds a threshold, $e_\Delta = P_{Write} - P_{Read} \geq 0.1$. The authors suggested two other indicators file deletion and file type funneling. File deletion is the removal of a file from the system; file type funneling checks for imbalances in the file types a process reads from and writes to. The authors did not provide details on how to use these two features in the scoring mechanism.

The authors observed that ransomware usually breaks each of the primary indicators' rules, while none of the benign applications did it. The IDS algorithm keeps a scoring system for each

process; when a process exceeds a threshold computed on the three indicators, Cryptolock flags it as malicious and raises the alarm or blocks it. This is an early-stop mechanism to prevent the ransomware from encrypting all the files; during the early phase of the attack, some files will be lost. This flexibility reduces the number of false positives and makes the system usable.

Cryptolock is implemented in two main parts: a kernel module intercepts I/O operations and notifies them to a second module, which analyzes them and computes the score. The authors developed a sandbox to perform experiments; the filesystem was populated using studies to examine the distribution of files over the filesystem and user document directories and using data from a collection of available real-world files for research purposes. The resulting directory tree was placed in the Cuckoo Sandbox [47], configured to disable the security measures. A sandbox [48] is a security mechanism for safe running and analyzing malicious software; it provides separation from the host system to avoid harm to the host machine. The malware sample was obtained from Virustotal [49], an online service for analyzing malware and downloading samples using ransomware-related keywords; a total of 2663 samples were acquired. The ransomware was checked to ensure detonation; as introduced above 2.1, ransomware may not start encrypting files after execution; for example, ransomware may try to connect to a [2] Command and Control (C&C) server for basic setup operations such as encryption key exchange. However, the ransomware may contain all the required components to detonate and start. The authors checked if ransomware encrypted the file by letting it run in the sandbox for 20 minutes and then checked if the files were encrypted. Of the initial group, 2171 samples were inert and were discarded, leaving 492 samples for the triage.

In terms of performance, Cryptodrop was able to detect all the 492 ransomware with a very small data loss. Given the behavior-related nature of the IDS, it is not possible to protect all the files; the median file lost was 10, with 33 as the worst-case scenario. The file system contains 5099 files, so the overall loss is minimal. Also, the main indicators proved valuable as 93% of the trial samples triggered all of them.

However, as suggested by the authors, the three indicators can be eluded by minor modifications of the attack. If ransomware encrypts the contents of a file without modifying the magic number, the indicator is eluded. Also, entropy and similarity values can be altered by inserting low entropy bits. Although not perfect in the long run, Cryptolock raises the bar in terms of the complexity required to launch an attack.

2.3.2 Redemption

Redemption [11] is a behavioral rule-based IDS similar to Crytolock 2.3.1; the author identifies a set of behaviors associated with ransomware encryption activities and defines rules to formalize them. The rules are aggregated to compute a malicious score S; each time a process asks for access to a privileged file, Redemption computes its malicious score. Suppose the score exceeds a given threshold α, Redemption flags it as malicious. In addition, Redemption has a module for fully recovering the lost files; in this work, we have considered how IDSs perform over time, so we focused on the IDS module. The authors based the rules on the observed ransomware behavior features. For each feature, the authors construct a rule for producing a score. Redemption aggregates the score using a weighted average and flags the process as malicious if the average exceeds a threshold. The author chose the weight by performing several experiments on the features. However, while Cryptolock monitors only file-modification operations, Redemption also considers how the processes behave. The author divides the features into two groups according to this distinction: content-based features and behavioral-based features. **Contents-base** features are entropy, file contents overwrite, and the delete operation; they focus on the file's modifications. **Process-behavior** features are Directory Traversal, Converting to a Specific File Type, and Access Frequency; they focus on how the process behaves.

Entropy is the same as in 2.3.1. For every read-and-write request to a file, Redemption computes the entropy of the corresponding data buffers. The main idea is to monitor if the file modification increases the entropy. For example, an ASCII file test has byte values between 0 and 127, but after encryption, the byte range occupies the whole spectrum, increasing the associated byte sequence entropy. If the write entropy is higher than the read entropy, the score is normalized in a [0,1] interval; if, instead, the read entropy is higher than the write entropy, the score is zero.

The **file content overwrite** checks the extension of the write operation over the total file size. When ransomware encrypts a file, it overwrites a large portion of its bytes. The score is computed as the total size of the data block modified over the file size, giving a value in the [0,1] interval.

The **file delete operation** is designed to capture the behavior of ransomware acting as Class C, which creates a new file with the cipher text and deletes the original one. If a process deletes a file, the score gets assigned a value of 1.

Directory Traversal checks on how many files a ransomware is performing writing operation; ransomware typically scans the file system and, when it gets the content of a directory,

starts encrypting all the files. File compression programs iterate over the contents of a directory to compress the files and save them in an archive, performing an operation with high associated entropy; however, the associated requests are for reading the files, not writing them. Redemption monitors write operations and consider the total number of privileged access over the number of files in the directory.

Converting to a specific file type controls imbalances in the extension of files written by the process. Authors classify two processes as belonging to different classes if they can access files of different extensions. For example, a PDF reader cannot open a Word document. Some processes combine together files of different extensions - as in a multimedia program - but they usually just read the file, not write them. Ransomware instead writes on files with many file extensions. If a process performs write operations on files of different extensions, the score is assigned a value of 1.

Access Frequency considers how frequent write operations are; ransomware attempts to quickly encrypt a large number of files; hence, the elapsed time between write operations will be very low. The frequency is computed as the inverse between the elapsed of two write operations. A speedy writing process will have a score close to 1. For the implementation, authors collected samples from public repositories [1] and blogs. A total of 9432 samples were collected, with 1174 encrypting-ransomware confirmed. For the Bening applications data, the authors recorded the operation of 5 users in the interaction with the file system. Authors compare the average malicious score for Benign applications and for the ransomware family. The median minimum and maximum scores for the Benign application are 0.027 and 0.0885, while for the ransomware, they are 0.43 and 0.73. Hence, in terms of scores, ransomware is one order of magnitude higher. The choice of the threshold α is a trade-off between detection and usability. A low threshold improves detection but increases the number of false positives and vice versa. The authors selected $\alpha = 0.12$ as it achieves the best detection and the false positive rates, with 100% accuracy and 0.5% false positive rate. The author used a test set for verifying the results, with a 100% accuracy and 0.5% false positive rate. The authors suggested two possible limitations of Redemption. A flagged ransomware may ask the user's permission to execute - for example, by a pop-up message complaining that the IDS is blocking it; if the user allows the ransomware, there is no way to stop it. The second limitation is related to the content-based features; an attacker may alter the ransomware file-related operations to lower the associated scores. The authors suggested three tactics. For entropy, use a low entropy payload; for file override, use selecting file overwrite; for file delete operation, launch periodic file destruction.

[1] At the time of writing, the two repositories seem no longer active.

The author estimated that Redemption would still capture iteration in file write operations.

2.3.3 RW guard

RWguard [5] is an IDS with hybrid characteristics; it has both ruled-based and behavioral-based models. The IDS is composed of 5 different modules: Decoy Monitoring, Process Monitoring, File Change Monitoring, File Classification, and CryptoAPI Function Hooking. An initial **parser** handles I/O operations and sends them to the relevant modules. The Decoy Monitor is standalone and checks encryption on decoy files. Process Monitoring and File Change Monitoring verify if a process is making anomalous modifications to files and signal it to the File Classification and the CryptoAPI Function Hooking for further checks. File Classification predicts the probability of benign file modification while the CryptoAPI Function Hooking checks if the user has legitimately encrypted the file associated with the alarm.

The **Decoy monitor** is rule-based IDS; it plans decoy files across the system and checks write operations on them. As any process should not modify such files' contents for any reason, monitoring them can tell that ransomware is acting in the system. It receives only write operations and checks if a decoy file is modified; in this case, it flags the process as malicious. The decoy files have common file extensions (txt, .doc, .pdf, .ppt, and .xls) to make a suitable target for ransomware.

File change monitoring is a rule-based IDS file oriented to monitor file changes. This module checks how the write operations modify a file. The authors identified a set of features related to ransomware activities and, for each of them set a rule according to the change on the file, specifying what is considered a legitimate operation. If the modification breaks one of the rules, the corresponding process is considered malicious. Unlike Cryptolock, just a single violation causes File change monitoring to signal the process to the Process and File classification modules. A **File manager** module stores the current properties of each file: type, entropy, size, last modification, etc. When a new file is created, the file manager adds a new entry with its characteristics. The file manager monitors the write operations and updates the file information. For each write operation on a file, the File change monitor uses the information in the file manager to verify if the write operation is considered malicious. The authors identified four features: similarity, entropy, file type change,

Similarity score is defined as in Cryptolock and provides a value between 0 and 100, which represents the statistical similarity between two files: 0 is a low correlation, like in a plain-text cipher-text pair, and 100 is a very high resemblance. Rwguard measures the similarity between

the same file before and after a write operation. A zero similarity score triggers the rule.

Entropy is defined as in Cryptolock and it measures the randomness in a byte sequence in a [0,8] range; encrypted files usually have entropy close to 8. Rwguard measures the entropy of a file after encryption, and if the value is above 6, the rule triggers.

File type change is defined as in Cryptolock; any kind of operation that modifies the file type - i.e., a change of its magic byte sequence - triggers the rule;

File size change check for modification in the file size; the rule triggers for a "significant file size change" - the authors did not provide a precise definition of "significant".

The **Process monitoring** is a behavioral-based IDS file-oriented with an ML model. The authors trained a supervised ML model as described in Section 2.2. The dataset is composed of ransomware and benign operations on file, which are used to profile legitimate and malicious processes. The ML model is a classifier that takes in input a series of operations, labeling them as legitimate or malicious. The Windows OS performs file operations by sending I/O request packets (IRPs). The features are the file operation performed to access and manipulate files: Write, Read, Open, Create, Close, and the number of temporary files.

Open is the request sent to the file system to access the file. The OS checks the permission and provides the process an *handle* to access the file.

Close is the opposite of open; the process notifies the OS that it no longer needs access to a file.

Read is the operation of reading the contents of the file from a device (like a hard disk) to the main memory (RAM).

Write is the opposite of read; it modifies the file contents saved on the device with byte in the main memory.

The **number of temporary files** refers to the file typically created by ransomware to hold data temporally while copying or removing the original files. The authors defined temporary files as those with the TMP extension.

The authors collected packets for several ransomware families and programs, labeling them into two groups. They considered the following models ML for the IDS: naive Bayes, logistic regression, decision tree, and random Forest. The author chose the random forest model, as it performed best in terms of accuracy and false positive rates.

The **File Classification Module** provides a secondary check on encryption of operations flagged as suspicious by the File change monitoring and Process monitoring module. The model learns the users' encrypting habits and classifies each encryption as legitimate or not. If an encryption is considered malicious, the CryptoAPI module performs a further check to

reduce false negatives.

The **CryptoAPI** model further checks potentially benign encryption to reduce the false positives. The module intercepts the CryptoAPI library functions to track legitimate encryptions.

To evaluate performance, the authors tested the system against several ransomware families. The first observation is the effectiveness of the decoy file; the authors conducted experiments both with and without this module. The average number of file system operations (open, write, read, close) was lower when the decoy file was included. This is because the system raises an alarm as soon as a single decoy is encrypted. However, the decoy mechanism does not work for all the families; three families (Mamba, Petya, and Matsnu) did not encrypt any decoy files, as they use a predetermined list of files. Therefore, the authors tested the IDS without the decoy, using the other four modules.

The average detection time for the main ransomware process was 3.45 seconds. Since ransomware may spawn subprocesses during its lifecycles, the authors also tested the time RWGuard required to identify all of them. Detecting multiple processes took longer, with the average time to identify all malicious processes run by a single ransomware being 8.87 seconds.

Process Monitoring was faster than File change monitoring. This is because Process Monitoring does not require file encryption to signal a malicious process; RWguard can block a process as soon as the process starts following the attack pattern. However, Process Monitor produces more false positives, as some benign applications may exhibit similar patterns.

2.3.4 ShieldFS

ShieldFS [10] is a behavioral-based IDS with an ML model. As described in Section 2.3.5 and Section 2.3.3, the IDS focuses on the file operations to detect ransomware. Additionally, ShieldFS includes two modules: a File recovery module and a Cryptographic module. The File recovery module restores encrypted files, but its functionality is outside the scope of this thesis. The Cryptographic module intercepts crypto API calls to detect ransomware; however, ransomware can circumvent this by using custom functions. To train and test the IDS module, the authors created a supervised dataset comprising malicious and benign processes. They collected samples of ransomware and recorded user sessions to build the dataset. The task is to classify a process as malicious or benign. As mentioned in 2.3.3, the dataset includes the file operations monitored using IRP packets.

The proposed approach has two main components: an overall model that verifies all file operations and a tiered system of models for nested control. The tiered system allows ShieldFS to

monitor both the short-term and long-term behavior of the processes and the system. Similar to Cryptolock, the primary focus is on the effects of encryption; however, while Cryptolock considers individual files, ShieldFS evaluates the broad filesystem. As a result, ShieldFS is agnostic to the infection propagation mechanism. For example, the authors claimed that the system is immune to process injection attacks. In a process injection attack, a malicious process corrupts the memory of another process, manipulating its behavior. With process injection, ransomware can modify other processes and coordinate encryption across multiple processes. The authors selected a set of features related to ransomware operations: file read, file write, file rename, file listing, file type coverage, and write entropy.

File Read is the number of read operations, as described in Rwguard, normalized by the total number of files in the system.

File Write is the number of write operations, as described in Rwguard, normalized by the total number of files in the system.

File Rename is the number of files renamed or moved, normalized by the total number of files in the system.

Folder listing refers to the folder-listing operation normalized by the total number of folders in the system. Ransomware uses folder-listing operations to identify the files in a directory and find potential targets.

File type coverage is the total number of files accessed, normalized by the total number of files having the same extension. Ransomware targets a specific set of file extensions and tries to access all files with those extensions. In contrast, benign applications typically access only a fraction of them.

Write entropy is the average entropy of the write operations. Entropy is measured as defined in Cryptlock, while write operations are defined as in Rwguard.

Each feature *counts* the number of elements of the category. For example, File Read is the total number of filesystem Read operations; features are also normalized using the total number of elements of the category. For example, File Read is normalized over the total number of files.

The authors create a **tier size system of models** which monitors a different number of files so that each model specializes. At the process start-up, the detector controls its behavior with the smallest size mode. A special overwatch model named *system centric* considers every file in the system. When the number of files exceeds a threshold, the detector performs two actions. First, it starts monitoring all the previously and newly opened files with a model trained for a larger file size. Second, it starts the smaller size model for the newly opened files. Hence, there are two running instances of the smaller model; one considering all the previously opened

files and another for the newly opened ones. So, a process is assigned to a classifier according to the percentage of accessed files; the percentage of accessed files is divided into units called ticks. Each time the percentage of files reaches a new tick, the mechanism described above starts. A process is flagged as malicious if at least one classifier of the same tick size agrees for K consecutive ticks. Each classifier is a random forest of 100 trees.

The authors trained ShieldFS using ransomware and benign applications. The mechanism is similar to the one described in Cryptolock 2.3.1; the authors downloaded and checked several ransomware samples and prepared a virtual machine to perform the experiments, obtaining 383 valid samples. The benign data was collected recording the actions of volunteers performing ordinary activities. The authors tested how ShieldFS performed out of a virtual machine; they cloned physical machine hard drive and use them to experiment on the detection mechanism. The authors tested three ransomware and ShieldFS was able to detect all of them. Finally, the authors tested ShieldFS on a test set of unknown samples. ShieldFS detected 298 samples out of 305. The top-tier process-centric model performed 95.2% of the detections; the incremental models were effective for a small fraction of ransomware performing code injection.

The authors discussed some possible limitations of the detector. Ransomware could avoid the tick mechanism by injecting a process that had already accessed a large number of files, assuming that the target process has not already been stopped. However, as soon as the target process started encrypting the files, ShieldFS would identify it. Ransomware may scale code injection and infect several processes, each of them performing a small fraction of the work. The authors claimed that multiprocess encryption is mitigated by the system-centric and incremental multi-tier strategy.

2.3.5 Unveil

Unveil [9] is a behavioral IDS rule-based, which monitors both Locker ransomware and Crypto ransomware. As explained in Section 2.1, this work focused on Crypto ransomware, so we considered only this functionality.

As in Crypolock, the authors identify patterns in the I/O file manipulation operations; accordingly to the authors, no matter how ransomware implements an attack, for example, the key management and generation, its action will follow under one of the proposed patterns. In the first attack pattern, ransomware performs a Read and a Write operation on the same file. Ransomware opens a file, reads the file's contents, encrypts it, writes the file, and finally closes it. In the second pattern, the ransomware manipulates three files. First, it gets the content of

			Model		
Feature	Cryptolock	Unveil	ShieldFS	Redemption	RWGuard
Entropy	X	X	X	X	X
Similarity	X				X
Delete	X			X	
File type change	X				X
File read		X	X		X
File write		X	X		X
File type conversion				X	
Override ratio				X	
Access frequency				X	
Directory traversal				X	
File type coverage			X		
File Rename/Move			X		
Folder listing			X		
File size change					X
File create		X			X
File close		X			X
Temporary Files					X

Table 2.1: I/O feature by detector

a file with a file Read operation. Next, it creates a new file and performs a file Write operation with the encrypted contents. To make the original file unavailable, it deletes the original file. However, the deleted files are not completely removed from memory, so they could be recovered without the key. The last pattern is a combination of the first two, the ransomware operates on three files. As in the second pattern, it reads a file and creates a new file with the encrypted version. Finally, it writes the encrypted content on the original file. With this strategy, also the original content is encrypted and so unrecoverable without the knowledge of the key.

2.4 Features Identified

From the review of the state of the art, we categorize all the features across the SotA models. Table 2.1 lists all the features and the models using them. The features in the table represent what a model uses, not how a model uses it. For example, both Unveil and ShieldFS use the file read operation, but in different ways; Unveil checks the file read to determine an access pattern, while ShieldFS normalizes it over the total number of files. However, as we discuss in

Section 3.2.2, the mechanism to collect them is the same. Entropy is the most popular feature and it is used by all the models. Indeed, with entropy, it is possible to determine how a process is transforming a file without knowing the exact sequence of operation. As suggested in [9], another possible detection tool is based on tracking the series of operations performed by a process, for example, using the API or the network activities. However, ransomware may try to hide their behavior to avoid detection. As suggested in 2.3.1, monitoring file activity has the benefit of being agnostic to all the previous activities. Both ShieldFS and RWGuard build ML learning models to train classifiers. Both use file and read operations to monitor file access. However, ShieldFS focuses on the broader picture of modifications, with features such as file type coverage and folder listing, which consider how a process behaves concerning the entire system. In contrast, RWGuard's ML model focuses on the modifications made by individual processes, such as how the process alters the size of the files or the number of temporary files created.

In the literature, we found additional models that build IDSs for ransomware [2]. All of these works use behavioral features, and some also integrate static features. Besides file operations, the two other most prominent feature types are API calls and registry operations. API calls allow monitoring how a process interacts with the operating system through its functions. Many operations can only be performed via the OS, so tracking the API calls executed by a process is an effective way to understand what it is actually doing. For example, reading a file will correspond to an API call to the OS's read function. In Windows, registry operations are used to modify the configuration of the operating system. As noted in [14], Wannacry modifies the registry to ensure its persistence. The models for IDS are either rule-based or behavioral-based, with the latter primarily relying on ML models. The most popular models used include support vector machines, deep learning techniques, and random forests.

2.5 Data-Shift problem

As introduced in Section 2.2, supervised ML models capture a relationship between a set of features X and a label Y. The dataset is split into two parts, one for training the model and the other for testing its performance. The underlying assumption is that the training and the test distribution are the same; however, data may not be stationary over time and space, leading to a degradation of machine model performance over time [50]. For the spam filter models, the shift in the features, the common characteristic in email sent, deteriorates the accuracy of the predictions with error rates growing from 5.95 to 48.81% [34].

The relationship between X and Y can be formalized as their joint probability $P(X, Y)$; the dataset shift occurs when the training and test joint distributions differ, namely $P_{tr}(X, Y) \neq P_{tst}(X, Y)$ [12]. With a dataset shift, the model learned from the training set does not perform well in the test set [51].

To study the dataset shift it is convenient to decompose the joint probability of the features and the label. The joint probability of two random variables A, B is symmetric, $P(A, B) = P(Y, X)$, and can be decomposed using the chain rule $P(A, B) = P(A|B)P(B)$. The term $P(A|B)$ is the probability of the event A knowing that the event B has occurred. In addition, it is possible to express the joint probability as $P(A, B) = P(B|A)P(A)$, where the conditional probability is computed knowing that A has occurred. Using these properties, we express the joint probability as $P(X, Y) = P(Y, X) = P(X|Y)P(Y) = P(Y|X)P(X)$. Hence, the joint probability can be rewritten in two alternative equivalent forms[52]:

- $P(X|Y)P(Y)$
- $P(Y|X)P(X)$

If one of these elements changes across the training $P_{tr}(X, Y)$ and test $P_{tr}(X, Y)$, a dataset shift occurs. The basic classification of dataset shift causes is named after the modification of a single element.

The **covariate shift** is the change in the distribution of the feature X, $P_{tr}(X) \neq P_{tst}(X)$, where the conditional distribution of Y over X remains the same, $P_{tr}(Y|X) = P_{tst}(Y|X)$. This happens if the features in the training and test have different distributions, while the relationship between X and Y does not change.

The **prior probability shift** is the change in the distribution of the label Y, $P_{tr}(Y) \neq P_{tst}(Y)$, where the conditional distribution of X over Y remains the same, $P_{tr}(X|Y) = P_{tst}(X|Y)$.

The **concept shift** covers two cases: the change in the conditional distribution of Y over X, $P_{tr}(Y|X) \neq P_{tst}(Y|X)$, where the distribution of X remains the same, $P_{tr}(X) = P_{tst}(X)$, or the change in the conditional distribution of X over Y, $P_{tr}(X|X) \neq P_{tst}(Y|X)$, where the distribution of Y remains the same, $P_{tr}(Y) = P_{tst}(Y)$. The concept shift occurs when the relationship between the covariates and the label changes over time, as in the email spam filters [53]. The cases introduced above are the most significant and observed in real-world datasets, so the other cases are generically referred to as "other types" of shift.

Dataset shift can emerge from significant events or natural evolution. The COVID-19 pandemic altered the medical landscape, introducing major disruption in routine care, changes in

medical treatment, and introduction of new vaccines [54]. The corps of half a million legitimate emails sent over five years in the energy company Enron [53] show several drifts in the frequency analysis of keywords related to the company's business activities. As we saw above, also ransomware had a natural evolution [17]. The encryption scheme became more sophisticated over time, from the use of symmetric encryption to mixed schemes with asymmetric encryption, where the attacker encrypts the key, making it more difficult to recover the data. This type of evolution is not strictly related to the activity of the IDS, as for the transition from locker-ransomware to crypto-ransomware. The second mechanism is more effective for blackmailing the victim, hence its increasing popularity. In the cybersecurity landscape, an attacker may modify its tactics with the precise objective of avoiding detection. In the case of the spam filter, the attacker will eventually observe that its strategy is no longer working and so will modify its tactics [34]. In the Hidden text salting, the attacker inserts hidden words, for example in the HTML tags, invisible to the user but able to alter the filter. In [6], the authors illustrate how an attacker may alter the ransomware to avoid detection, for example, by inserting low entropy sequences to reduce the overall entropy.

The dataset shift is a common problem, so also the datasets used for training and testing ransomware IDS may be subject to it. We observed that the experimental settings of research work on ransomware IDS development follow a similar pattern: the authors obtain a set of samples, discard the non-encrypting ones, and use them as a base to train a model. As we saw in Section 2.1, ransomware is organized in families, so authors expose results by family 2.3.4 2.3.3. However, authors make no distinction when dealing with the time-related aspects; as we saw in Section 2.1, a ransomware family is a building block for creating ransomware; so different attackers can take the same family and derive ransomware with different characteristics. Thus, the release of ransomware also has a temporal component, for example, due to the ongoing activities of old and new criminal groups. The Virustotal database collects samples of malware and records also time information like the first submission date [49]. So, even within samples of the same family, there could be some differences. The resulting dataset may suffer from the data shift problem, as the resulting patterns incorporated by the trained models have inconsistent characteristics.

3
Framework Design

In this chapter, we provide the motivation for the work, the proposed framework, and the results obtained by using it. Section 3.1 explains the need for a framework for ransomware data collection. During this work, we discovered that data shift is not the only bias that may emerge; there is also a bias related to the availability of families, leading to an overrepresentation of one ransomware family over another. In Sections 3.2, we present the proposed framework for collecting the data. First, we introduce the mechanisms for capturing ransomware I/O operations. Next, we illustrate the three main components: the sandbox for executing ransomware, the software for data collection, and the detonation scripting for performing the operations. In Section 3.3, we describe how we obtained the ransomware samples and the challenges that we faced during the testing process.

3.1 Motivation

In this section, we emphasize the importance of establishing a clear framework for data collection and categorization of ransomware across both family and temporal dimensions. As discussed in Section 2.5, the data shift problem can deteriorate the performance of IDS over time. In the ransomware literature, authors often claim high accuracy rates in detection. However, if the proposed IDSs are not periodically checked over time, the accuracy is not guaranteed to be the same. On top of that, the model may be trained on samples not equally distributed over time, and the ransomware samples are not typically categorized over time. As we discuss in

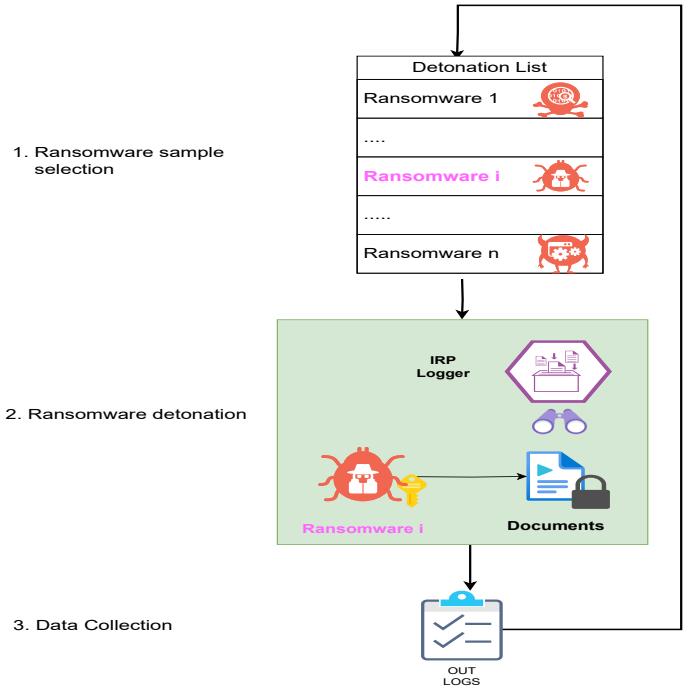


Figure 3.1: Data collection mechanism: A ransomware is selected from a list; next it detonated in the sandbox where its actions are recorded; the logs are extracted and the process restarts with another ransomware.

Section 3.3, collecting ransomware samples is not easy. Moreover, researches are limited by the ransomware available in the repositories. If a period is overrepresented or underrepresented, little can be done to correct it. Table 3.2 shows that ransomware samples are not equally spread across the years. As we discuss in Section 3.3, time is not the only factor to consider. When checking for detonation, we noticed that some ransomware families were harder to detonate than others, as shown in Table 3.3. The BlackCat family, for example, requires an access token to function, adding an additional barrier to detonation. In one case, we could not detonate any sample from the Hive family. In contrast, the Phobos family detonated easily. This introduces a bias in the model, as certain families are excluded. Another crucial aspect is the clear identification and categorization of the ransomware samples used. To foster comparability of results, models should be compared using the same ransomware.

3.2 Proposed framework

Due to the large volume of data required, both in the number of families and samples, the data collection process requires a mechanism to automatize it. Figure 3.1 provides an overview of the data collection process. The process begins with a list of ransomware samples, each of which is executed in a sandbox environment to record filesystem operations during execution. The resulting logs are extracted and saved. This mechanism enables the automatic retrieval of logs detailing the operations of a given group of ransomware.

The three main components of the data collection process are the sandbox, the IRPLogger, and the detonation script. The sandbox is the environment for safety running the ransomware; after each detonation, the sandbox is restored to its previous state to nullify any effect of the ransomware. The **IRPlogger** is the software tool to record the file operations, capturing details such as the timestamp, process ID, and the type of operation in a log file. The detonation script coordinates all the activities, eliminating the need for manual intervention. The log files are named with a timestamp and the SHA256 of the ransomware allowing for flexible grouping in multiple ways. Online repositories like Virustotal [49] or MalwareBazaar [55] provide API for detailed descriptions of the ransomware characteristics, such as the first recorded appearance of the ransomware or the imported libraries.

Figure 3.2 provides an overview of the packet capture in the sandbox. When ransomware attempts to access a file, it cannot do it directly; instead, it must send a request to the file system. The file system processes the requests and prepares a request data packet for the hardware components storing the file. At the same time, it forwards a copy of the data to the IRPLogger. The IRPLogger observes the requests, processes them, and saves the details to a log file. It does not interact directly with the ransomware but merely monitors and records the file access requests.

IRPlogger is the software used to capture file operations; it intercepts these operations, extracts the required information, and saves it to a file. Figure 3.2 provides an overview of how the IRPLogger captures the details of these operations. In Windows OS, file operations are executed through a system data structure named I/O request packet (IRP). The IRP packet contains all the relevant details, such as the operation type, the target file, and operation flags. Figure 3.3 illustrates the capture of a read request to a .pdf file named REPORT.pdf. The file system prepares an IRP packet with all the details and forwards it to the required file. If the IRP logger is active, a copy of the IRP packet is sent to the IRPLogger, which then processes the packet's field and records an entry for the logs.

In the next sections we provide additional detail of each of the components. First, we ex-

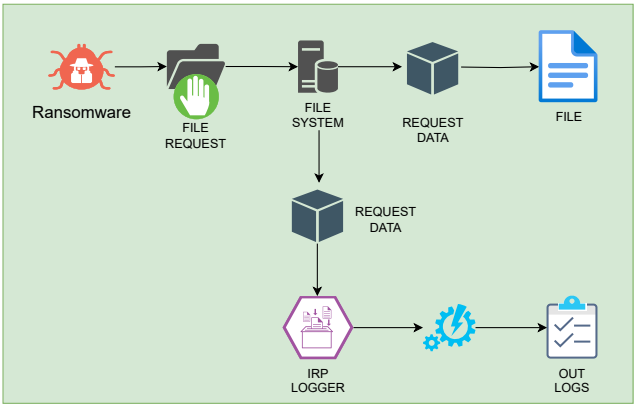


Figure 3.2: Capture overview: The ransomware interacts with the file system to encrypt the file. The file system forwards the request to the IRPLogger, which processes it and saves it to a file.

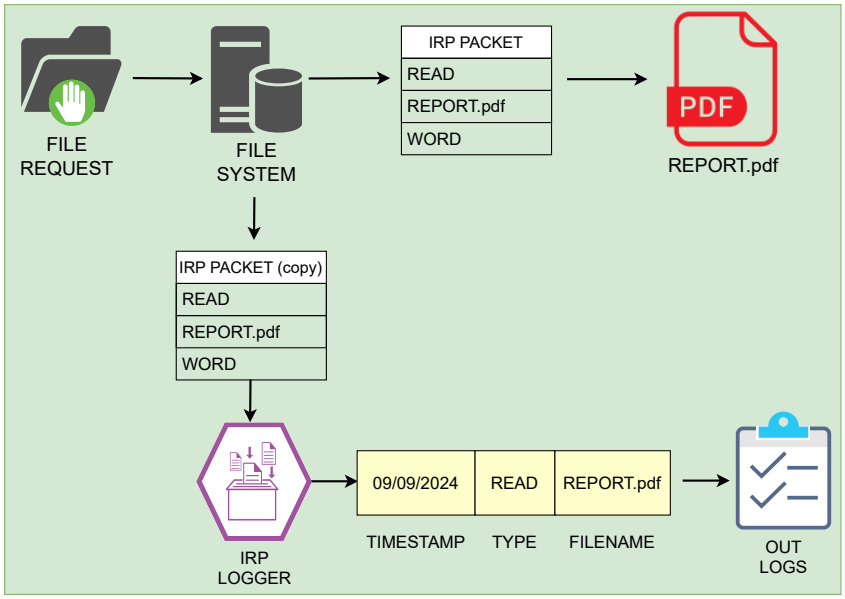


Figure 3.3: IRPLogger overview: for each file request, the file system sends a data structure named IRP packet with all the details of the operation. The IRPLogger receives a copy of the packet, parses its content, and saves it to a file.

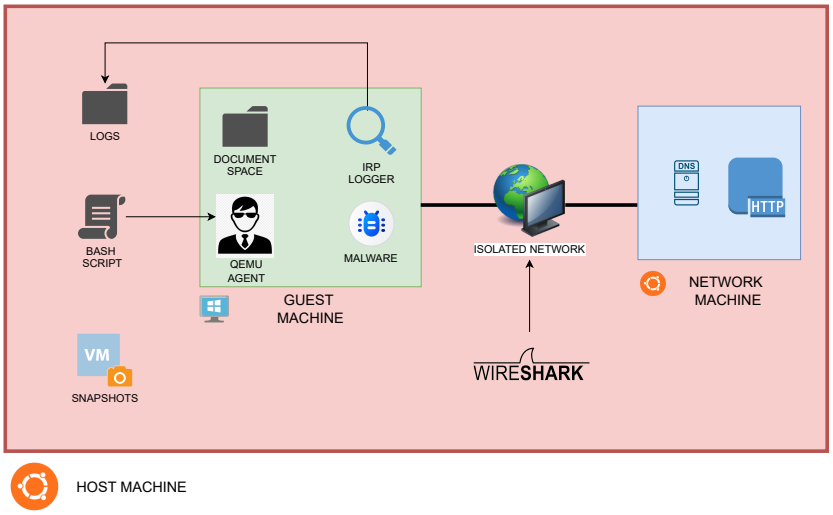


Figure 3.4: Sandbox organization: the Ubuntu host machine uses virtualization to run a Windows guest machine. The guest machine detonates the ransomware and collects the logs of the operations using the IRPLogger. A third virtual machine is used to simulate the Web.

plain how the sandbox operates and how it detonates ransomware. Next, we offer a more detailed illustration of how the IRPLogger functions within the sandbox. Finally, we describe the detonation script used to coordinate the detonation process and the IRPLogger within the sandbox.

3.2.1 Sandbox

The sandbox is the safe space for running ransomware and recording their actions. Figure 3.4 shows an overview of the proposed sandbox. The host machine runs all the components, while the guest machine is used for the detonation. Additionally, a network machine simulates web services, such as web servers and DNS servers.

The **host** machine is the physical computer that runs all the components of the data collection process. It operates the virtualization software necessary to run the guest machine. Virtualization emulates a computer system by simulating all its hardware and software components, allowing multiple types of software to run on a single machine while keeping the virtual machines strongly isolated from the host machine. The host machine stores the virtual machines used for executing the ransomware, saves the logs of operations, and executes the detonation script to initiate ransomware. It is also responsible for storing snapshots of the guest machine. A snapshot is a file that captures all the details of a virtual machine at a specific moment, includ-

ing its configuration and files, making it possible to revert a machine to its previous status. By maintaining these snapshots, the host machine can easily reset the environment for repeated testing. Furthermore, the host machine simulates an isolated network to enable network operations between virtual machines.

The **guest** machine is a virtual machine used for the detonation of ransomware samples. During ransomware execution, it runs the IRPLogger software to capture file operations. It also includes a software component that facilitates communication with the host machine, enabling the guest to receive messages, report its status, execute processes, and simulate hardware operations such as mouse movements and keyboard typing. These capabilities are utilized by the detonation script to automate the entire process.

The **network** machine is a virtual machine used to simulate network operations. As illustrated in Section 2.1, ransomware may contact a remote machine during the attack. The network machine simulates both a web server and a DNS server, providing standardized responses. The DNS server simulation returns the IP address of the network machine, redirecting web requests to it. The web server then provides a simple web page in response, simulating a successful connection. This machine is connected to the guest machine through an isolated network, ensuring complete isolation. Additionally, it is possible to capture the network traffic of the isolated network using Wireshark.

Ransomware detonations occur within the Guest machine; hence, ensuring its isolation is critical to prevent the spread of the malware. This setup guarantees the following safety measures:

1. The Host machine is Ubuntu while the Guest is a Windows 10, reducing the spread surface. Linux and Windows systems have structural differences, so cross-operating system infection is very different.

2. Detonation is executed on a virtual machine using the KVM hypervisor. The virtual machine has its kernel, and it is isolated from the Host machine, making the spread very difficult.

3. The Guest machine runs on an isolated network.

This setup guarantees the reproducibility of the experiments; it is sufficient to provide a copy of the Guest machine's disk and the bash script. Besides the installation of QEMU-KVM, the Host machine does not have any peculiar setup.

Operation	Timestamp	PID	Major Operation	Entropy	Filename
IRP	18:05:28:766	5912	IRP_MJ_READ	1.82	report.pdf
FIO	18:05:28:875	6016	IRP_MJ_WRITE	2.877	report.pdf

Table 3.1: Example of logs saved by IRP with some columns omitted for clarity.

3.2.2 IRPLogger

The IRPLogger is the software component that captures the flow of IRP packets. Table 3.1 provides an example of some of the fields saved by the IRPLogger (with some columns omitted for clarity). The IRPLogger records the type of operation, IRP or fast I/O (FIO) [56]. FIO is a fast operation designed for rapid synchronization. The timestamp field indicates the time of the operation, while the PID represents the process ID of the process making the request. The major operation field specifies the type of operation, such as read or write. The entropy field is Shannon's entropy associated with the buffer used for the operation. The filename is the name of the targets of the operation.

The IRPLogger is an expanded version of the IRPLogger used in Minerva [41], which is derived from the original Microsoft Minifilter driver [57]. Third parties can implement "minifilter" drivers that interface with the OS, intercepting all the I/O requests.

Figure 3.5 presents a simplified diagram illustrating the organization of the IRPLogger. The IRPLogger consists of two main modules: the kernel and the user modules, named after the address space in which they operate. The user space is a separate address space of a regular process, while the kernel space is a special address space where privileged instructions are executed. When a process needs to access a file, it sends a file request to the I/O manager. The I/O manager then forwards a copy of the request to the kernel module of the IRPLogger. The kernel module parses the contents of the IRP packet contents and stores the parameter in a custom data frame. This data frame is then sent to the user module, which further parses its contents and saves them in a log file.

In Section 2.4, we listed all the required features to train the SotA IDSs. To capture these features, we expanded the original IRPLogger. While the IRPLogger includes additional components for data frame exchange between the kernel and the user modules (other than the ones shown in Figure 3.5), these components are not involved in saving the contents of the IRP packets. Thus, we focused on enhancing the kernel and user components responsible for processing and saving the IRP packets, adding additional rules depending on the IRP packet type.

To identify the features of interest, we needed a thorough understanding of how IRP pack-

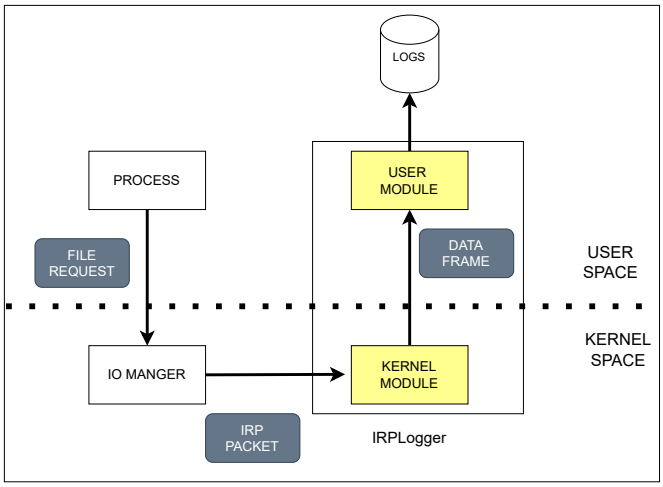


Figure 3.5: IRPLogger organization: the kernel module receives IRP packets from the OS and packs the parameters in a custom data frame. The user module saves the contents in a log file.

ets are internally organized to correctly configure the kernel module. Figure 3.6 illustrates the basic structure of an IRP packet [58]. An IRP packet consists of two main sections: the header and the I/O stack locations. The header is the fixed part of the packet that the OS uses internally to process the packet. The I/O stack location contains the parameter of the request and is the component that the IRPLogger parses to extract the details of the operation. The io stack location has two main fields: the major function code and the Parameters. The major function code is a unique identifier that specifies the type of operation such as read or write. The Parameters field is a union of structs that holds the actual contents of the request. Figure 3.6 illustrates the *representation* of the packet, not its actual contents. At runtime, the OS processes the file requests by creating an IRP packet with the appropriate fields. Therefore, the actual content of the Parameters will be the struct containing the parameters of the operation.

The value in the parameters field depends on the type of operation. Figure 3.7 highlights the difference between the IRP packet for the write operation 3.7a and the IRP packet for the rename operation 3.7b. In a write IRP, the Parameters field contains a pointer to the buffer that stores the bytes to be written to the file. In contrast, a rename IRP includes a pointer to the string representing the new name of the file.

The amount of processing required to extract each feature varies depending on its nature. For example, obtaining the number of write operations is straightforward: it involves counting the labels with the value IRP_MJ_WRITE from the logs of the operations. However, comput-

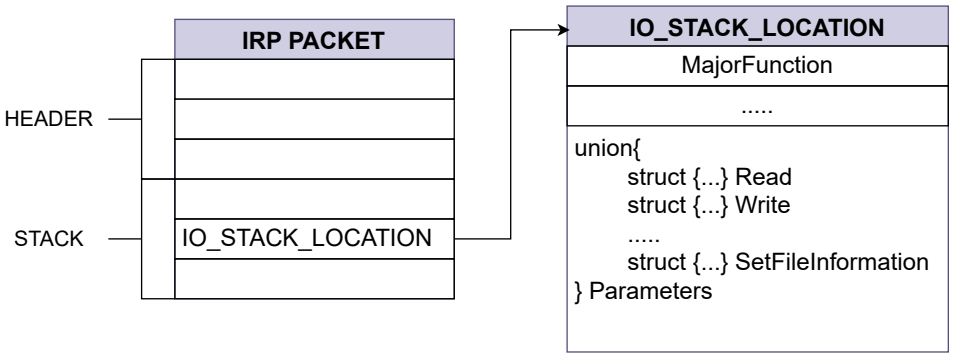


Figure 3.6: IRP Packet basic representation: the stack portion of the IRP packet contains the data for the file operation in the io stack location. The io stack location is represented by a Parameters union which contains the details

(a) IRP packet for write operation

(b) IRP packet for rename operation

Figure 3.7: Comparision of the IRP packets for the write and rename operation: the fields in the parameter struct are operation-specific. The write operation has a pointer to the buffer with the byte for the write operation, the rename a pointer to the string with the new name.

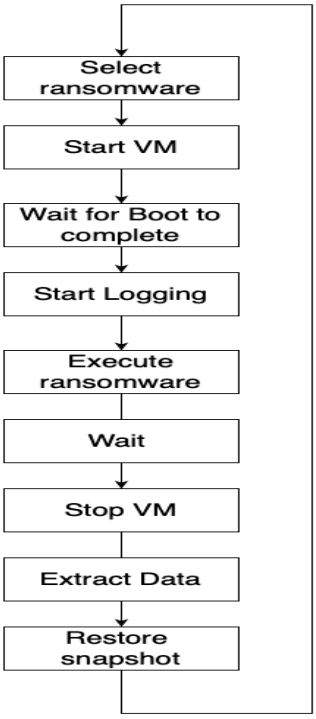


Figure 3.8: Detonation script workflow: steps for detonating ransomware in the sandbox and collecting the data.

ing the entropy associated with an operation, as shown in Figure 3.7a, requires the IRPLogger to access the file corresponding buffer and to apply the entropy formula to its bytes. Similarly, to retrieve the new file name after a rename operation, the IRPLogger must copy the string containing the new name, Figure 3.7b. Other features can be computed from the log files. For example, the access frequency which measures the time between two write operations, can be calculated using the time stamps of the write operations. Section 4.2 provides more details about how the operations are performed.

Based on this understanding, the IRPLogger was modified in the following way: In the kernel module, we introduced additional rules based on the major function code and possible file rename values. In the data frame used for communication between the two components, we added a union of structs to include additional parameters. The user module was modified to parse the additional fields and save them to a file.

The detonation script is a bash script in the host machine. The script coordinates the entire detonation process. Its main functions are to boot the guest machine, start the IRPLogger, detonate the ransomware, extract the data, and then revert the guest machine to its previous

state. Figure 3.8 illustrates the flow.

3.2.3 Detonation script

The detonation script iterates over a list of ransomware samples, all of which are pre-installed on the guest machine. Section 4.1.1 describes the program used to execute commands on the guest machine from the host. For each sample, the detonation script performs the same set of operations. First, it starts the guest machine from a dedicated snapshot for the detonation. Section 4.1.2 provides details on how the guest machine configuration.

Since the virtual machine does not clearly delineate between hardware setup and operating system startup, the script waits for a signal from the guest machine to ensure that the OS is fully ready before proceeding. The detonation script spawns a console that listens for incoming text strings from the guest machine. Upon startup, the guest machine sends a specific text message to indicate its readiness, as detailed in Section 4.1.2. The detonation script pauses and waits for this message, continuing only after receiving the signal.

This approach is crucial because starting subsequent steps only when the guest machine is fully loaded prevents commands from being executed prematurely. If commands are sent too early, they may not be recognized or accepted by the guest machine, leading to a failed detonation.

Once the guest machine is ready, the script loads the IRPLogger module and begins logging operations. To prevent the ransomware from encrypting the log file, the file is stored in a directory containing configuration files that the ransomware is programmed to ignore.

All ransomware samples are already present on the guest machine, compressed in ZIP format with a standard password for security. The detonation script invokes the 7zip program to extract the ransomware.

To execute the ransomware, two techniques are employed: using the QEMU Guest Agent or simulating a user typing commands. The QEMU Guest Agent requires the executable's path to start the ransomware. Alternatively, the script can use the *send-key* program to simulate a user typing the necessary commands on the keyboard. The script opens a Windows terminal with administrative privileges and invokes the ransomware from it. Section 3.3 describes why this second procedure was necessary.

A timeout is set to allow the ransomware to perform its activities, such as identifying and encrypting files, which can vary in duration depending on the specific ransomware's behavior. When the timeout ends, the script powers off the guest machine.

The script extracts the log file from the guest machine. To avoid opening a direct communication channel between the host and the guest, such as an SSH connection, the log file is extracted directly from the disk image of the guest machine. The script uses the *virt-copy-out* tool, which extracts a file from a disk image by simply providing its path. This ensures that the ransomware is not running during extraction. Finally, the script reverts the virtual machine to its previous state and moves on to the next sample.

3.3 Ransomware Sample

In this Section, we explain how we obtained the ransomware samples and how we used the detonation mechanism to test them. The section shows the main challenge we faced during the work and shows that obtaining a balanced dataset is not an easy task.

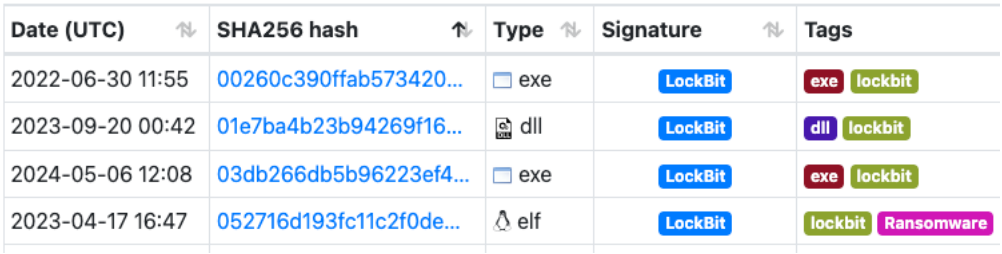
3.3.1 Sample acquisition

To collect samples, we used the Malwarebazaar online repository[55]. Malwarebazaar enables users to search for and download malware samples from a centralized source. The browse page allows users to search through the available samples, and an API is provided to download JSON files containing metadata about these samples. This metadata is crucial for correctly classifying ransomware and organizing research work efficiently.

The browse menu on the Malwarebazaar webpage allows users to inspect the available samples. Figure 3.9 illustrates some of the fields that are available on the browse web page. The SHA256 field represents the hash of the ransomware code and is used to uniquely identify a ransomware sample. The Type field specifies the file type; as shown in the figure, ransomware samples are available in Windows executable (exe), Dynamic Link Library (DLL), Linux Executable, and Linkable Format (elf). The Signature field indicates the ransomware family, while the Tags field contains additional information that helps classify the ransomware.

Malwarebazaar's API [59] allows users to query the available catalog of malware and download samples. By using the API, it is possible to set filters and obtain a list of malware in JSON format. We searched for popular ransomware family signatures and downloaded a JSON file containing a list of matching ransomware samples.

For each ransomware sample, the JSON file provides various details, including the file type, the first-seen date, and other relevant characteristics. As shown in Figure 3.9, there are several file formats, such as EXE, DLL, ELF, and more. We chose to download executable (.exe) files,



Date (UTC)	SHA256 hash	Type	Signature	Tags
2022-06-30 11:55	00260c390ffab573420...	exe	LockBit	exe lockbit
2023-09-20 00:42	01e7ba4b23b94269f16...	dll	LockBit	dll lockbit
2024-05-06 12:08	03db266db5b96223ef4...	exe	LockBit	exe lockbit
2023-04-17 16:47	052716d193fc11c2f0de...	elf	LockBit	lockbit Ransomware

Figure 3.9: Malwarebazaare browse menu allows to search for malware in the repository. The SHA256 hash is used as an identifier. The signature is the estimated file member, the type is the file type, and the tags are strings with additional information to classify ransomware.

Family	First seen year					TOT
	2020	2021	2022	2023	2024	
Blackbasta			20	3	1	24
Blackcat		5	54	15		74
Cerber	68	1	8	8		85
Chaos				90	3	93
Conti		6	37	35		78
Crysis	36			1		37
Hive			83	6	1	90
Lockbit			3	46	23	72
Phobos			22	59	13	94
Ryuk	17	9	5	8	2	41
Wannacry			18	56	6	80

Table 3.2: Subset of available .exe ransomware samples from the Malwarebazaar repository. The year reference to the date of first submission on the platform.

as they can be directly executed on a Windows machine, making them suitable for our purposes. Table 3.2 presents a subset of all available ransomware samples available in the repository, selected using Malwarebazaar's *Tag* criteria to identify the ransomware family. The "first year of submission" is determined based on the date the sample was uploaded to the platform. After obtaining the samples, we began analyzing them to determine how many could successfully detonate. In Section 3.3.2, we explain the criteria and methods used to assess whether a sample was suitable for our investigation.

3.3.2 Detonation

Detonating an executable in a sandbox does not always guarantee a successful outcome. Using the downloaded samples, we began investigating their behavior within the sandbox environment. The goal of this process was not to collect data but to determine whether a sample could be used for further analysis. Therefore, we manually launched ransomware samples in the sandbox and observed their behavior in real-time. We faced two main challenges: determining which ransomware could actually detonate and how to document and register a detonation.

To document the detonation of a sample, we employed the detonation mechanism described in Section 3.2.3. However, determining a detonation from the logs proved challenging, as the OS performs hundreds of operations in the background. Therefore, identifying ransomware successfully encrypting the files from just the logs proved a time-consuming process. To make things faster, we modified the snapshot used for the detonation, inserting **bait** files in the most common directories checked by ransomware: the desktop, the user directory, the program directory, and the program 86 directory. In each of them, we placed a directory with files with the *.txt* and the *.jpg* extensions, as they are common ransomware targets. We executed the detonation for a reasonable period and then extracted the directories with the bait files using the same mechanism used for the logs. Figure 3.10 shows a successful detonation of a sample of the ryuk's family.

During the detonation, the ransomware renamed all files in the directories, appending the *.RYK* extension and adding the *RyukReadMe.html* file. The *out.log* generated by the IRP logger, captures details of the I/O operations performed. The *metalog.md* file collects logs of the detonation operations and records any errors encountered during the process. This setup allowed us to quickly determine which ransomware samples had successfully detonated by examining the presence and content of these log files after each test.

During the manual detonation of a sample, we encountered several issues:

- Ransomware not modifying the file extensions
- Ransomware not running
- Ransomware running but not encrypting
- Ransomware deletes itself without encrypting
- Ransomware checks for an online resource
- Ransomware check for not existing online resource

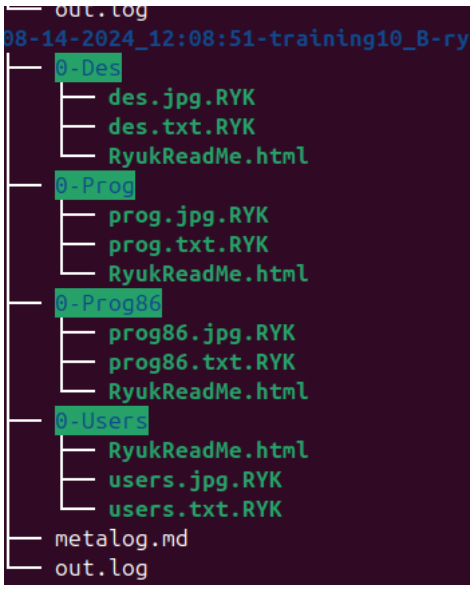


Figure 3.10: Successful detonation of a sample of the Ryuk family. Ransomware encrypted the contents of all the bait files and added the extension.RYK. In addition, It added payment instructions in all the directories.

- Ransomware requests additional parameters for detonation

Figure 3.11 shows another example of why using just the logs to detect a detonation was difficult; the ransomware encrypted the file without modifying the extension, and the only sign of detonation was the presence of payment instructions.

Ransomware may not run due to an error in the software, a missing library, or a lack of a resource. If the ransomware starts normally, the guest machine returns the process ID of the running process. However, if the ransomware fails to detonate, the guest machine sends an error message, reporting the failure. The error message is saved in the metalog.md file of the detonation. Figure 3.12 illustrates an example of a failed detonation message from the guest machine. Section 4.1.1 explains how the host machine receives an error message from the guest machine. To further investigate the error, it is possible to manually execute the ransomware on the host machine, as shown in 3.13. In this case, the Windows error message confirms that the ransomware could not be loaded. Since there was no feasible solution for these samples, they were discarded.

Even if the ransomware starts successfully, it may still fail to encrypt files or perform only partial encryption. To identify such cases, we monitored the ransomware's behavior in the sandbox using the tools available within Windows. The Task Manager allowed us to see the details of the running process, as shown in Figure 3.14. In this example, the ransomware process

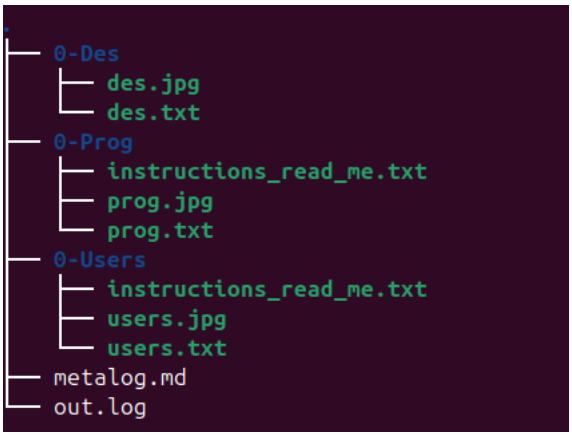


Figure 3.11: Detonation without file extension modification

```
run malware wannacry 413d0c3bc23ed24cca0c1eca9680bb820738e0549f3913a6ad8426e18275199a
error: internal error: unable to execute QEMU agent command 'guest-exec': Guest agent co
mmand failed, error was 'Failed to execute child process (Invalid argument)'
```

Figure 3.12: Logs of ransomware not running

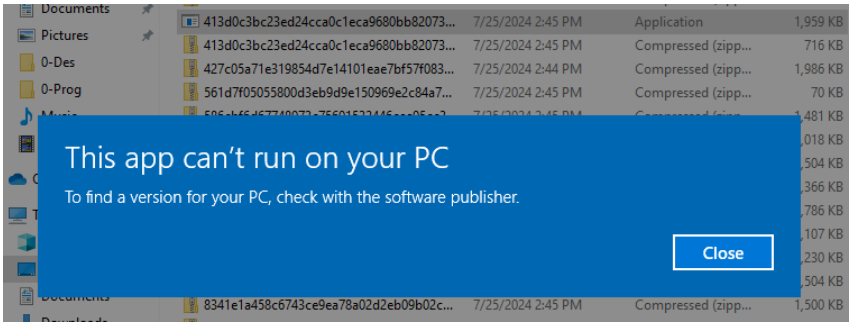


Figure 3.13: Ransomware not running

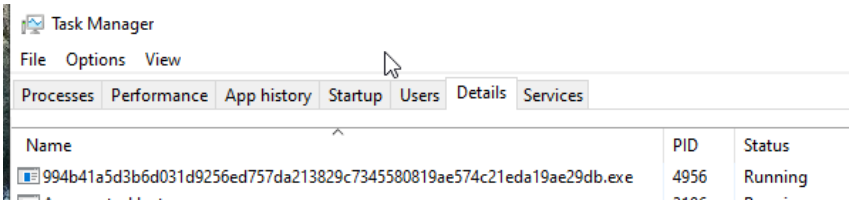


Figure 3.14: ransomware running without encryption

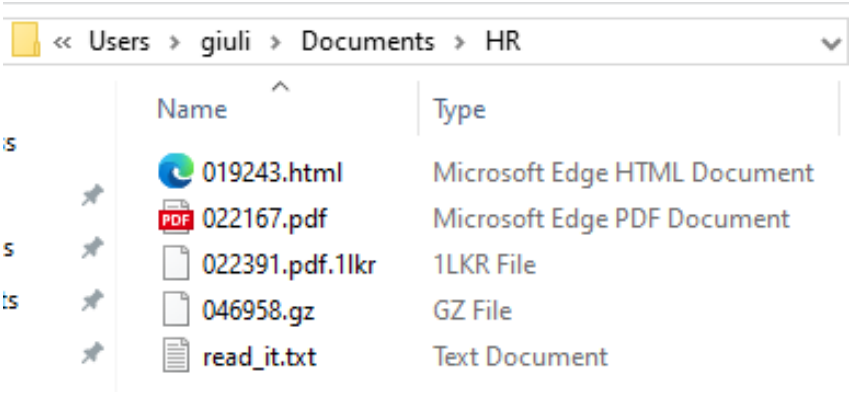


Figure 3.15: Partial encryption

is running, but no files are being encrypted. Since our objective is to capture the encryption behavior, such cases are considered ineffective and are not useful for our analysis.

Ransomware may only partially encrypt the files, as shown in Figure 3.15. The ransomware added the payment instruction but encrypted only one file with extension `1LKR`. Analyzing such situations in detail would require a static analysis of the executable to understand why only partial encryption occurred. Since this goes beyond the scope of our dynamic analysis and would require additional resources, we did not include these samples in our study.

Ransomware may delete itself without detonating. During detonation testing, we observed that certain ransomware families—such as Cerber, Crysis, and Chaos—did not detonate when executed using the QEMU-Guest-agent from the host machine. However, they could be activated successfully by executing them from the PowerShell terminal within the guest machine as the logged-in user. As introduced in 3.2.3, we simulated user keystrokes to run the ransomware, as shown in Figure 3.16. This behavior appears to be influenced by the user running the process. Using the Task Manager, we checked the user associated with the process. When using the QEMU-Guest-agent, the user is run as SYSTEM, and in this context, these ransomware families do not initiate encryption. In contrast, when the ransomware is executed by the logged-in user on the Powershell terminal, the ransomware detonates correctly. To handle these cases, we

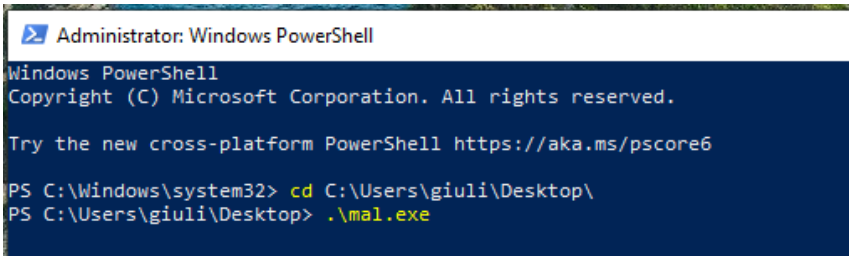


Figure 3.16: Manual detonation

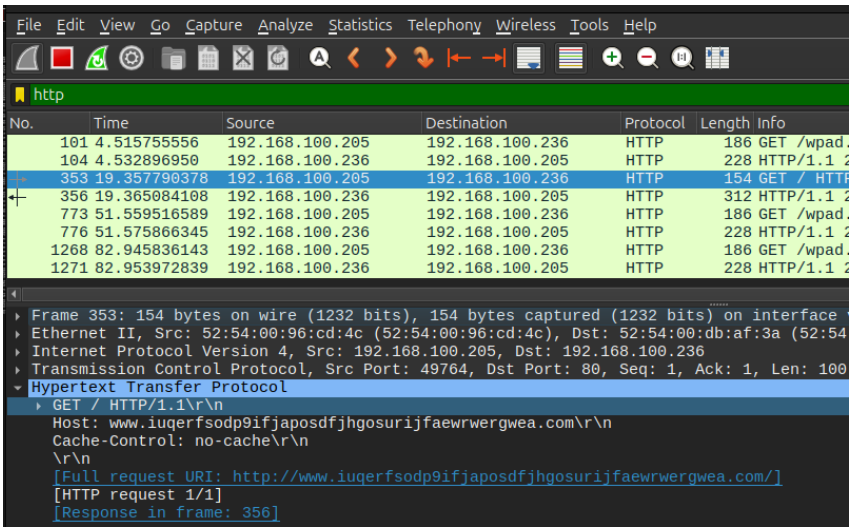


Figure 3.17: Wannacry requesting URL

employed a small library to simulate keystrokes on the guest machine's keyboard, allowing us to mimic the actions of a logged-in user, as described in Section 4.1.1.

As described in 3.2.1, the sandbox includes a network component to simulate Web requests, enabling us to test ransomware that checks for online services before detonating. Ransomware may perform these checks to confirm it is not running in a sandbox environment. Wannacry is a corner case. As shown in Fig 3.17, it performs a web request, but if it gets a response, it does not detonate. Hence, we had to turn off the network simulation for this family.

We also tested ransomware samples that had previously failed to detonate by re-running them with network simulation enabled. The idea was that simulating network activity might trigger ransomware checks during the deployment phase, as described in Section 2.1. However, the network simulation did not significantly improve the detonation of additional samples.

Ransomware may require additional parameters for detonation. For example, the Blackcat ransomware requests an initial access token to begin its detonation, potentially as a tactic to

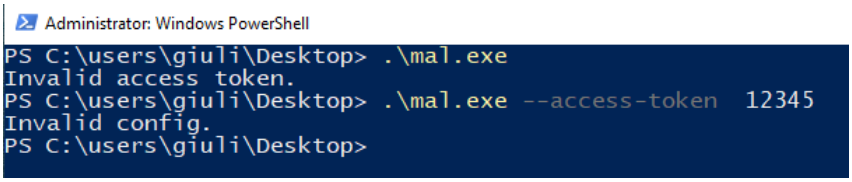


Figure 3.18: Blackcat invalid access token

Family	First seen year					TOT
	2020	2021	2022	2023	2024	
Blackbasta			17	3	1	21
Blackcat		3	11			14
Cerber	40	1	2	3		46
Chaos				60	3	63
Conti		4	14	9		27
Crysis	35				1	36
Hive			0	0	0	0
Lockbit			2	22	11	35
Phobos			19	56	7	82
Ryuk	10	4	4	4	1	23
Wannacry			2	9	3	14

Table 3.3: Number of encrypting ransomware samples by family and first seen year

slow down the work of analysts. To address this, we examined a known access token referenced in technical reports and used it during the detonation testing to attempt to trigger the ransomware's behavior [60]. However, the required token can vary across samples, leading to a failed detonation if the correct token is not provided, as shown in Figure 3.18.

Table 3.3 shows the successful detonations by the ransomware family. For the Hive family, we could not achieve a single successful detonation. We inspected the directories with the bait files, but they did not show any renaming, nor did the directories contain payment instructions. We also tested the files to check whether the ransomware had encrypted them without changing the extensions, but the files were still regularly accessible.

Some families exhibited higher success rates; for example, the Phobos family produced consistent results over the years, as did the samples in the Crysis and Cerber families. The BlackCat family, however, proved more challenging to test due to its access token requirements. The Wannacry family proved difficult to obtain, resulting in a small number of detonations.

The results highlight the overall difficulty of conducting research on ransomware. The first challenge is obtaining samples that are evenly distributed over time. For example, the Cry-

sis ransomware shows results for only 2020 and 2024. The second challenge is making the ransomware actually execute. As discussed in Section 3.3.1, determining how to trigger ransomware can be quite complex. During the detonation testing, the ransomware exhibited various behaviors, including different ransom notes and several screen-locking messages. Providing a single framework for analyzing all these cases is not easy. Another problem is related to the OS used. Initially, we used Windows 11; however, to verify cross-compatibility, we also tested on Windows 10 and noticed an increase in the number of successful detonations. Therefore, a potential future direction could be testing the detonation of the same ransomware across different versions of the target OS. Another challenge is the classification and dating of ransomware. The first year reported in the table is when the sample was provided to the repository. The underlying assumption is that, in the early stages, someone would encounter the ransomware, detect it, and upload it to the repository. However, there is no guarantee that this behavior always occurs, and the actual release date may differ. Another element to consider is the file type used. In this work, we focused on .exe files, but this is not the only type, as ransomware can also come in DLL form.

4 Implementation

This chapter provides additional details on the implementation of the framework and the internal mechanisms of the IRP (I/O Request Packet). Section 4.1 describes the main components of the sandbox and how they are configured to ensure the proper functioning of the detonation mechanism. These components interact with each other and must be capable of communicating and coordinating their actions effectively. Section 4.2 reviews the structure of the IRP packet and I/O operations in the Windows OS, serving as a reference for understanding how these operations are transformed into features.

4.1 Sandbox components

This section provides details of the sandbox and its components, including the host machine, the guest machine, the network machine, and the isolated network. For each component, we highlight its basic functionality, its relationship with the other components, and its role in the detonation mechanism.

4.1.1 Host machine

The host machine is central to the sandbox environment, serving as the platform that runs the guest machine and executes the detonation script. Figure 4.1 shows an overview of its structure.

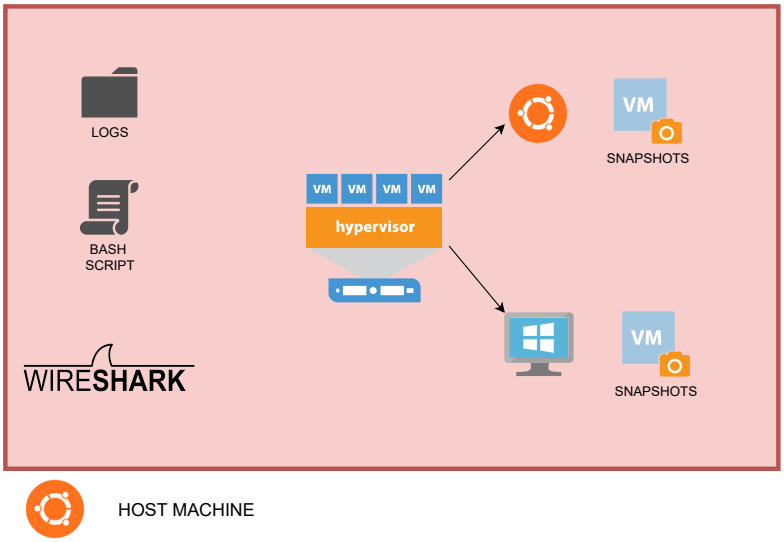


Figure 4.1: The host machine runs the virtualization software for the virtual machine and contains the snapshots and the detonation script. It also contains the logs extracted from the detonations.

Virtualization is executed using the *QEMU-KVM* software stack. In this stack, **KVM** functions as the **hypervisor** module. A hypervisor is a software layer that creates and manages virtual machines, providing a fully isolated environment separated from the Host machine. While virtual machines (VMs) typically have a lower performance compared to containers (such as those managed by Docker), they offer two significant advantages first, each VM runs its own simulated kernel, whereas a container shares it with the host. Second, containers limit the OS choice to those compatible with the host's kernel, while a hypervisor allows running any operating system. This flexibility makes it possible, for example, to run a Windows VM on a Linux host

QEMU is an open-source emulator that enables the simulation of complete computer systems and various architectures. It is complemented by **Libvirt**, an open-source API and management tool designed for handling virtual machines. Libvirt offers two main interfaces for managing VMs: a graphical user interface called **virt-manager** and a command-line interface named **Virsh**. These tools facilitate monitoring the status of virtual machines and executing commands within them.

The detonation script described in Section 3.2.3 leverages Virsh commands to launch virtual machines and execute specific programs within them. Virsh communicates with the qemu-agent-command installed on the guest machine to carry out these tasks; further information

about the guest-agent software can be found in Section 4.1.2. When a program is executed on the guest machine, it returns either the process ID (PID) of the program or an error message, allowing the host to monitor the execution status. The detonation script uses this command to initiate the IRPLogger, as well as to unzip and detonate the ransomware.

The Host machine executes all the operations using **bash** scripts to minimize the configurations and actions needed on the guest machine. Modifying the Guest machine can be a time-consuming process, as it requires reverting to a previous snapshot and then re-saving it. In contrast, changes to the host machine are much more straightforward, involving only updates to the script files.

To perform some of the detonations, we simulated a user typing commands on the keyboard. The Virsh s*end-key* command is used to send keyboard-like operations to the guest machine, with each key represented by a specific byte code. To facilitate this process, we developed a small library capable of executing tasks such as opening PowerShell in administrator mode and typing strings. The library accepts a string as input, converts it into the corresponding bytes sequence using an array, and sends these bytes to simulate key presses. To ensure all keystrokes are registered accurately, the function introduces small pauses between key presses to prevent any from being missed. This approach allows us to automate interactions with the Windows environment, such as opening the Run dialog with the "Win" + "R" combination or launching PowerShell with administrative privileges using "Ctrl" + "Shift" + "Enter."

Each virtual machine can save its current state in a series of **snapshots**. A snapshot captures the exact state of a virtual machine at a particular moment, enabling the machine to be reverted to this state as needed. Additionally, snapshots can be branched, allowing for different testing scenarios from a common baseline.

4.1.2 Guest machine

The guest machine is tasked with executing all detonations and running the IRPLogger. Within the guest machine, a program called QEMU-Guest-agent runs continuously, waiting for instructions from the host machine and executing them accordingly. Upon completing its startup process, the guest machine also sends a notification to the host machine, confirming that it is ready to proceed with further operations. Figure 4.2 shows an overview of its components;

The QEMU-Guest-agent is a helper tool installed on the guest machine that facilitates data exchange between the host and guest machines and enables command execution on the guest. The host machine sends JSON-formatted data to the guest machine, which the QEMU-Guest-

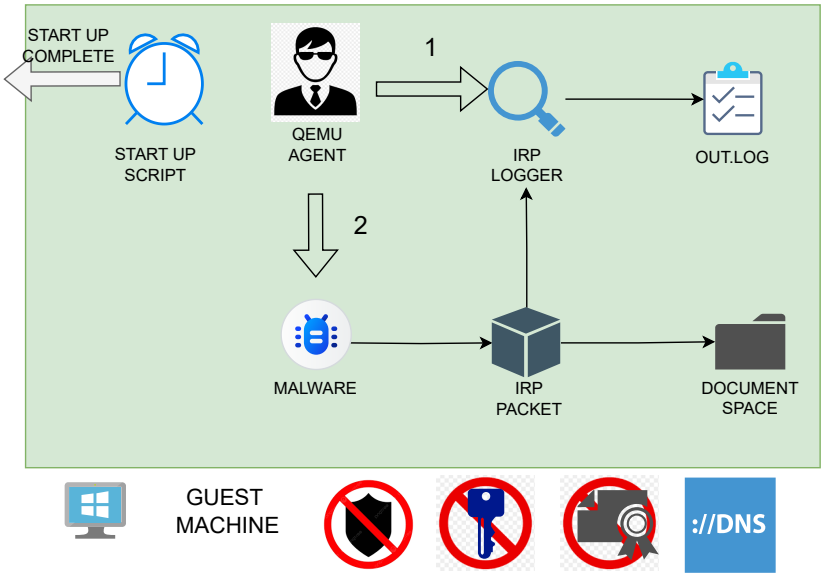


Figure 4.2: Guest machine overview. The QEMU-agent starts the IRPLogger (1) and detonates ransomware (2). Additionally, the guest machine sends a startup message to the host machine to notify that is ready. At the bottom of the figure, the additional settings of the guest machine: disabled window Defender, disabled password login, disabled driver signature, and custom DNS address.

agent parses and executes. These JSON strings specify the action to be performed, the path of the executable, and any additional arguments to be passed. The QEMU-Guest-agent uses this information to load and run the IRPLogger. The detonation script then selects the ransomware sample based on its SHA hash from the available options. The ransomware samples are preloaded into a specific directory on the guest machine. The guest machine first extracts the chosen ransomware sample and then executes it. Communication between the host and guest occurs through a dedicated channel, which could potentially present an attack surface for malware. However, for a malware attack to succeed, it would need to be specifically designed for a Windows OS environment and capable of infecting a Linux OS, which is a highly unlikely scenario. Upon detonation, the ransomware starts encrypting the contents of files on the guest machine. Each file operation triggers the generation of an IRP (I/O Request Packet). The IRPLogger intercepts these packets and records them in the ‘out.log‘ file, as detailed in Section 3.2.2. This log file is stored in the ‘C:__Windows‘ directory, which ransomware typically avoids encrypting to prevent rendering the system unusable. Furthermore, several files and directories were added to the environment to simulate real-world conditions, as described in Section 4.1.3.

To notify the detonation script that the guest machine has completed its startup process, we added a script on the guest machine that sends a message to the host upon completion. This script transmits a simple text message with the value "STARTUP_COMPLETE" to the host machine via a dedicated communication channel. To implement this, a link to the script is placed in the shell:startup directory. In Windows 10, any script in this directory is automatically executed once the user's startup process is finished. Consequently, when the guest machine is fully initialized, the script runs and sends the "STARTUP_COMPLETE" message to the host machine, indicating that the guest is ready for further operations. As described in Section 3.2.3, the detonation script continuously listens on the dedicated channel for this message and proceeds only after receiving it, ensuring that all subsequent actions are performed only when the guest machine is fully prepared.

The guest machine is configured with the following additional settings: disabling Windows Defender, disabling Driver Signature Enforcement, removing the login password, and setting the IP address of the DNS server. Microsoft enforces the Driver Signature Enforcement policy for all drivers that interact with the kernel, disallowing self-signed drivers. To be officially signed, all drivers must undergo a review process on the Microsoft platform. For the purposes of development and data collection, we disabled the driver signature enforcement policy on the guest machine. To allow the detonation of ransomware, we also disabled Windows Defender Antivirus on the guest machine; otherwise, the operating system would block the malware upon execution. This is a reasonable assumption for our work, as we are focusing on building the last line of defense, making this configuration choice without loss of generality. Additionally, we enabled automatic login without requiring a password. This simplifies the process by minimizing the necessary typing and reducing the number of steps required for verification. Without this configuration, extra checks would be needed to handle the login screen prompt and ensure the password is typed correctly. In Section 4.1.4, we describe how we configured the DNS settings for the guest machine.

4.1.3 Document space

To make the detonation environment realistic, the contents of the guest machine should closely resemble those of a real-world machine. A simple ransomware detonation is insufficient for our purposes, as we aim to collect data to understand how ransomware would behave in a typical machine. For example, we want to create a tree-like directory structure filled with files to observe how ransomware explores and interacts with the system. The two main components of

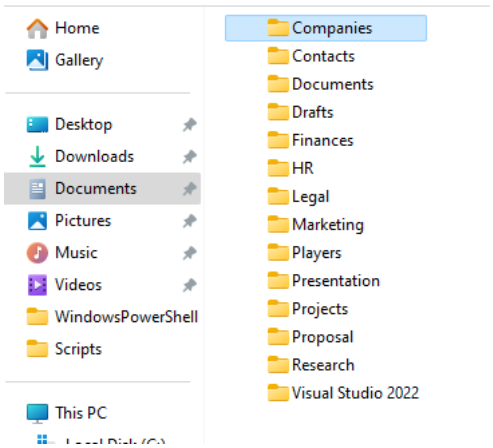


Figure 4.3: Document space first level

these contents are the files and directories, collectively referred to as the machine's document space [9].

To obtain actual files, we utilized the Govdocs1 dataset [61], a corpus of real files available for research purposes. To create the file tree structure, we used Impression [62], a tool designed to generate realistic file trees with multiple subdirectory layers and evenly distribute files among these subdirectories. However, Impression has two limitations: first, the files it generates are random blobs of data without any meaningful content; second, the directory names are assigned as numerical codes, which do not reflect a typical real-world environment.

To build the document space, we first ran Impression to create the skeleton of the directory structure. Next, we employed a bash script to replace each fake file with a real one from the Govdocs1 dataset. Finally, to make the directory names more realistic, we substituted the numerical codes with more familiar names, such as "Project" and "Research." Figure 4.3 displays the directories at the first level of the document space.

This procedure can be reused to create various document spaces, allowing us to collect data in different experimental settings.

4.1.4 Network

To enhance the realism of our environment, we added a third machine to simulate network and web services. Figure 4.4 provides an overview of this additional machine, referred to as the Network machine.

For the Network machine, we used the REMnux virtual machine[63]. REMnux is a toolkit

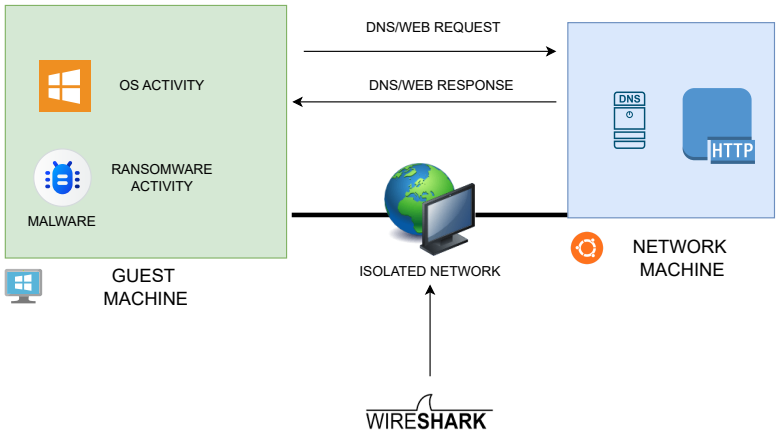


Figure 4.4: Network machine overview: the network machine provides simulation of the DNS and Web servers requests from the process in the guest machine. The two machines are connected in an isolated network. To capture the traffic, it is possible to use Wireshark on the isolated network.

designed for malware analysis. In this setup, we employed the *Inetsim* for simulating the ordinary network responses. Inetsim allows us to mimic DNS and Web request behaviors, as illustrated in Figure 4.4. On the guest machine, we set the preferred DNS IP address to point to the network machine's IP address, as shown in Figure 4.5

Figure 4.6 illustrates an HTTP request made from the guest machine within the isolated network. The process begins with a DNS request for the domain "example.com," followed by an HTTP request to the corresponding IP address. It is important to note that the destination IP address of both the DNS and HTTP requests is the same, as a single machine (the Network machine) provides both services.

This network and web service simulation adds a layer of realism to our environment, allowing us to observe how ransomware interacts with network resources, such as DNS resolution and HTTP requests, in a controlled, isolated setting.

4.2 I/O OPERATIONS BACKGROUND

In this section, we review the structure of the IRP packets and how Windows OS performs file operations.

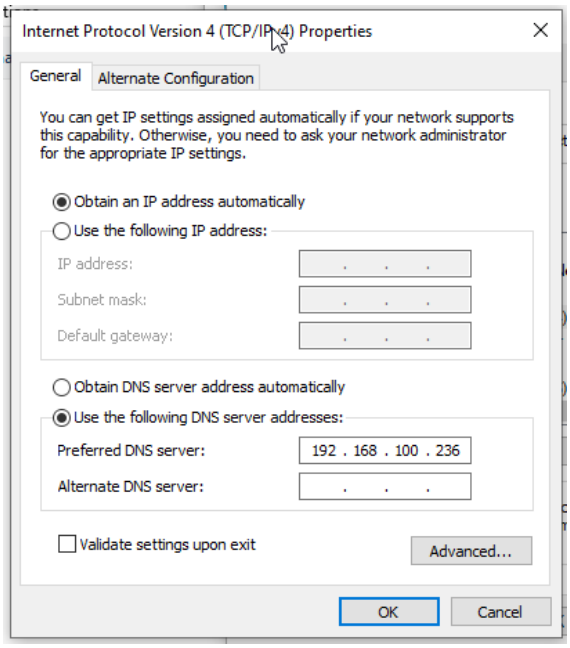


Figure 4.5: DNS settings in the Guest machine: the DNS IP address is manually set to the network machine

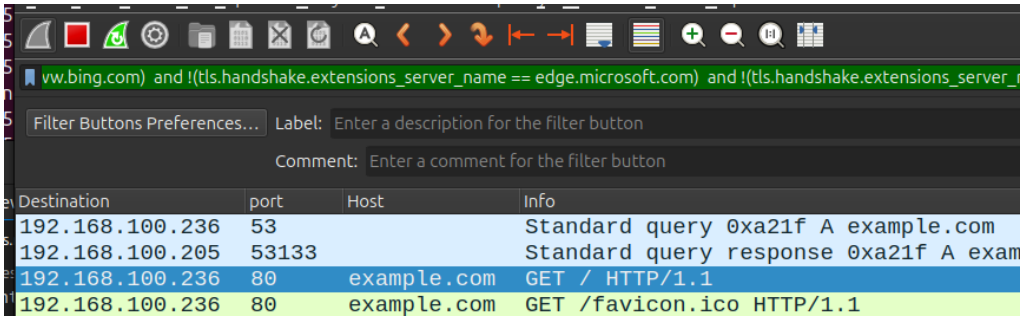


Figure 4.6: HTTP and DNS Simulation: the Wireshark capture shows the response of the DNS and Web server for the example.com URL. Both responses have the same IP address, as the network machine is providing both services.

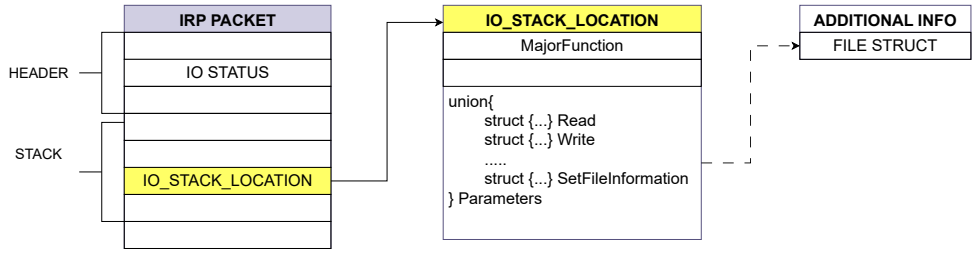


Figure 4.7: The IRP packet has a header and an io stack location with the operation parameters. The operation is characterized by a major function code (a unique identifier) and a Parameter variable; the Parameter is a union of structs that takes value according to the major function code. Some IRP packets may have additional parameters saved in an additional structure, as indicated by the dashed line.

4.2.1 IRP Packet

Inside Windows, the IRP packet is represented by an IRP structure. The IRP structure is a C -language structure that represents an I/O request packet. All the details of the operations are represented using C language primitive types (char, int, pointer, etc.) and data structure (struct and union). The primitives store the basic elements of the operation, for example, the ID number to identify an operation. The struct allows to combine in a single variable several elements related to the same operation. The union allows for some flexibility, as it allows the placement of different data types under the same label; at run time, the variable will contain only one of them. To provide more readable names to the data type, Microsoft uses the C language typedef functionality for creating multiple labels for the same data type. All the data types can be nested to construct more complex data types. To parse an IRP packet, it thus necessary to know its internal structure.

Figure 4.7 shows the representation of an IRP packet. An IRP packet is organized into two components: a header and an I/O stack location [64]. The I/O manager uses the header to store information about the operation, such as the drivers related to the operation. The I/O stack location contains the instructions for the driver, indicating what kind of operation should be performed. Some IRP packets may also include additional parameters, which are stored in a separate structure to provide further details required for the requested operation, represented by the dashed line in the figure.

The contents of I/O stack location IRP packets vary depending on the type of operation performed. Every packet has two key variables: the major function code and the Parameters. The **major function code** [65] is a variable of type UCHAR that specifies the type of operation requested. Table 4.1 reports some examples of major function codes and associated operations.

Major Function Code	Operation description
IRP_MJ_CREATE	request to open a handle to a file object
IRP_MJ_READ	transfers data from device to system memory
IRP_MJ_WRITE	transfers data from system memory to device
IRP_MJ_CLOSE	closed the file object handle

Table 4.1: Example of IRP operations

Structure Name	Description
Create	FLT_PARAMETERS for IRP_MJ_CREATE
Read	FLT_PARAMETERS for IRP_MJ_READ
Write	FLT_PARAMETERS for IRP_MJ_WRITE

Table 4.2: Example of Structure for IRP packet

The Parameters is a variable of type FLT_ PARAMETERS [66] which defines the request-type-specific parameters associated with an I/O operation. The FLT_PARAMETERS is a union of structures, where each structure corresponds to the parameter required by an IRP packet. The actual structure used at runtime depends on the type of IRP operation being performed. As an example, Table 4.2 provides the FLT_PARAMETERS structures for some major function codes. The IRP_MJ_CREATE major code has a structure named Create, the IRP_MJ_READ a struct named Read, and so forth. Thus, when an operation such as reading a file is requested, the Parameters variable will use the Read structure from the FLT _PARAMETERS union, which contains all the necessary details and attributes related to the file read operation.

Therefore, when opening a file, the relevant IRP parameters will be stored in the Create struct.

Figure 4.8 illustrates the IRP packet of a write operation. The major function code is IRP _MJ_WRITE; the Parameters variable, in this case, is of type Write and contains all the necessary information to modify the bytes in the target file. This includes a pointer to a buffer that stores the data to be written, the offset in the file where the write operation should begin, and the number of bytes to be written.

Some operations may require additional parameters for performing operations. Figure 4.9 illustrates the organization of the IRP packet containing rename information. In this case, the additional information is the new name of the file. The Major function code for this operation is IRP_MJ_SET _INFORMAITON, which is used to manipulate the metadata of a file, such as its name or position. The Parameters variable is a structure of type SetFile-

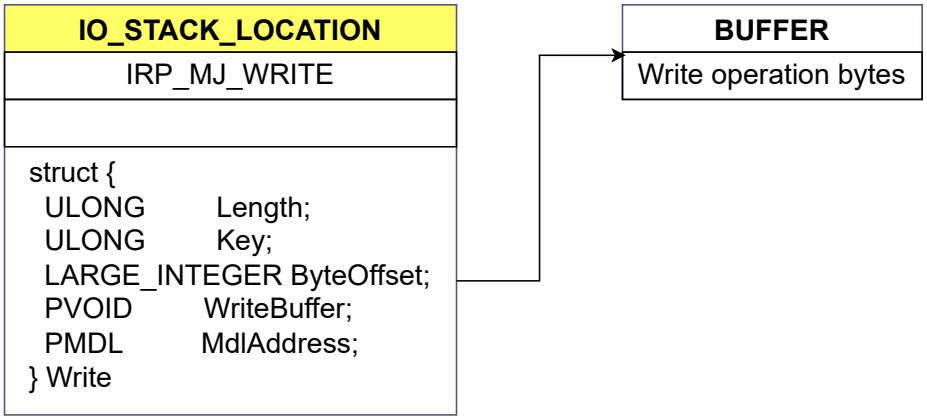


Figure 4.8: Complete IRP packet for write operation. The Write structure contains a pointer to the buffer with the byte to be written and also the offset where to write them on the file.

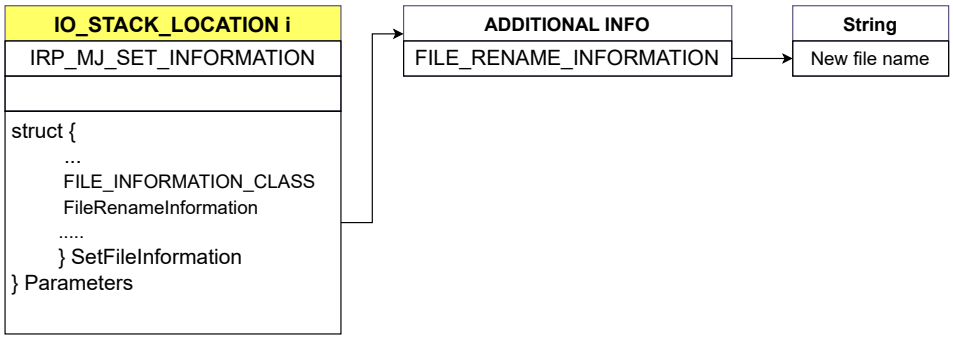


Figure 4.9: Complete IRP packet for rename operation. The SetFileInformation structure contains a pointer to a structure named FILE_RENAME_INFORMATION; the FILE_RENAME_INFORMATION indicates the name of the pointed structure.

Information. To indicate the presence of additional information, this structure includes a FILE_INFORMATION _CLASS variable, which is an enumeration. For the rename operation, the value of this enumeration is FileRenameInformation. The additional information required for the rename operation is packed in another structure of type FILE_RENAME _INFORMATION, which contains a pointer to the string representing the new file name.

The FILE_INFORMATION_CLASS [67] is an enumeration that specifies the type of additional parameters an IRP packet is carrying. Each value in this enumeration corresponds to a specific kind of operation, and for each value, there is an associated structure containing the actual parameters needed for that operation. Table 4.3 shows an example of elements in the FILE_INFORMATION_CLASS enumeration.

Name	Integer value
FileRenameInformation	10
FileFullDirectoryInformation	2
FileBothDirectoryInformation	3

Table 4.3: Example of file information class value

Enumeration Value	Associated struct
FileAllocationInformation	FILE_ALLOCATION_INFORMATION
FileBasicInformation	FILE_BASIC_INFORMATION
FileDispositionInformation	FILE_DISPOSITION_INFORMATION
FileEndOfFileInformation	FILE_END_OF_FILE_INFORMATION
FileLinkInformation	FILE_LINK_INFORMATION
FilePositionInformation	FILE_POSITION_INFORMATION
FileRenameInformation	FILE_RENAME_INFORMATION
FileValidDataLengthInformation	FILE_VALID_DATA_LENGTH_INFORMATION

Table 4.4: File information class values for IRP_MJ_SET_INFORMATION

The FILE_INFORMATION_CLASS enumeration provides a general reference of all possible values to be referenced, but usually, the IRP packet only uses a subset of them. As we have seen above, The OS sends an IRP_MJ_SET_INFORMATION request to set metadata about a file or file handle [68]. The OS reuses this packet for several operations, specifying a different type pair of enumeration/structure for that particular operation, as shown in Table 4.4.

4.2.2 I/O Management

An I/O operation is a general term for the communication between a computer and an external component, such as another computer or a device [69]. This category applies also to file-related operations. Indeed, the manipulation of files by a process is organized as an I/O operation. As part of their ordinary activities, processes perform operations on files to access their contents and/or manipulate them. A user process can import a file for direct manipulation, for example in Word, or another process may request to access a configuration file. Other processes may create files and write on them to record their operations, like writing logs or error messages. The file mechanism extends also to other components in the system, like a device or volume. However, users' processes are forbidden by the OS to directly access files. Indeed, the process performs a request to the OS, which, behind the curtain, arranges all the necessary preparations to provide access to a file. For example, assume a process wants to access a file in the hard drive.

If the system did not have an OS, the process would need to know how to interface with the particular model. A driver is a software component to access specific hardware and may vary across hardware producers. When accessing a hardware resource, the driver works as an API for directly accessing it. So to access a resource, a process would need what is the correct driver and know to invoke its function. To simplify things, the OS provides two services: it manages the driver and it invokes them at run time. The OS scans the system and lists the devices, and then it assigns to each of them the correct driver. For security reasons, the OS may also verify the origin of a driver by means of its associated digital signature [70]. A signature [71] is a digital certificate that uses cryptography to prove the authenticity of an entity, in this case, the manufacturer of the driver. When the process asks for a file, the OS intercepts the request and forwards it to the correct driver.

Hence, the OS provides several services to the process for performing low-level operations. Windows OS has special functions called Application Programming Interface (API), also known as syscalls in Linux, which the process can invoke to perform file-related operations. For example, the Windows API [72] provides a collection of functions for several tasks, from the data access store to the network. Hence, a human programmer, when writing software, can rely on them. For example, to open a file, a process specifies a file path and invokes the API function for opening a file. The name of the function for opening a file in Windows OS is *CreateFileA* [73]. The OS returns a special object named *handle* which the process can invoke to access the request.

On the user side, a process at run-time just has to invoke the function associated with the operation on a file. On the system side, however, there are several steps. When a process runs in a system, it is said to run in user space, while all the OS activities instead run in kernel space [74]. Kernel space is a memory area where privileged instruction can be executed, in contrast to user space, which has restrictions. When the process invokes the API, the execution goes into kernel space for further processing. The Windows kernel-mode I/O manager[75] manages the communication between applications and the interfaces provided by the device driver. Due to the difference in processing speed of the OS and the driver, the communication between the two components is done through I/O request packets (IRPs). The IRP packet is a standardized format for communication between the OS and the driver and between drivers. Drivers are organized in stack structures that parse the IRP packet contents, but the full description is beyond the scope of this thesis. The key point is that for every I/O operation, the OS sends a corresponding IRP packet. Hence, to collect the file-related operations, it is sufficient to intercept the IRP packets and save them.

5 Conclusion

5.1 Contributions

This work has reviewed the state-of-the-art in ransomware IDS research. Firstly, we examined the current state-of-the-art IDS approaches for ransomware detection. The primary focus of existing research is on detecting ransomware by monitoring file operations, as these directly reflect the final impact of the malware. Two main IDS models are used: rule-based and behavior-based. The rule-based model establishes fundamental rules to define the expected actions of ransomware and provides a scoring system to measure the severity of its activities. On the other hand, the behavior-based model builds a profile to establish a baseline of normal behavior, allowing deviations that suggest malicious activity to be detected. Both models rely heavily on historical data for their creation and evaluation. However, little to no attention has been paid to the temporal component of these models, raising concerns about how their performance may deteriorate over time.

To address this problem practically, we proposed a framework for data collection and analysis of the data shift problem. This framework automates the collection of ransomware data and the construction of a dataset. Given a group of ransomware, the mechanism executes them one by one, collects and extracts the operation performed, and restores the VM to its previous state. This allows to scale the collection of mechains providing a large amount of data. The framework's components can be modified for further testing or data collection. For example,

the file and directory in the guest machine can be modified to test how ransomware performs in different settings.

5.2 Lesson learned

During its development, we encountered several limitations in handling ransomware samples. Determining whether a ransomware sample is suitable for analysis is not straightforward, as each malware run must be thoroughly tested. Some of the issues could be addressed by modifying how the ransomware detonates. For the Cerber, Crysis, and Chaos families, we could increase the detonation by simulating a user typing the command on the keycode. The network machine did not prove helpful in increasing the number of detonations, as its use did not alter the behavior of the ransomware. An additional challenge is that the data collection mechanism is time-consuming. The virtual machine must be booted before starting the test and then shut down afterward. Additionally, the data must be extracted, and the snapshot must be reverted to its previous state.

As observed in this work, some ransomware families are more challenging to track than others. For instance, the BlackCat family requires a token to deliberately slow down research activities, while in other cases, such as with the Hive family, we were unable to trigger the ransomware to detonate. Additionally, the number of samples that actually perform encryption varies significantly across different families. For example, the WannaCry family has a relatively low number of samples, while the Phobos family has a much higher number.

5.3 Future works

As described above, we encountered several challenges during development that could serve as potential topics for further research. The ransomware samples tested were only of the .exe type, but online repositories provide other types, such as DLLs, ZIP files, or Office documents. Expanding the pool of available ransomware samples by including DLLs and other types could be beneficial. Another possible area of investigation is the analysis of failed detonations in ransomware. It could be beneficial to determine whether a ransomware sample fails to run due to a configuration problem, such as a missing library or other components, or because of a built-in kill switch. Indeed, for some ransomware samples, we received error messages indicating that the ransomware could not be loaded. In other cases, the ransomware ran in the background but did not encrypt any files. Testing each model to determine how the features change over time

is another area for expansion. The rule-based model can be very effective, but if the reality it models changes, the rules may no longer effectively protect the system. By using the proposed dataset, the model could be trained and evaluated to quantify how the data shift problem impacts performance. Another possible improvement is testing different versions of the operating system. In this work, we used Windows 10 to run ransomware, but it is not the only available system. Windows 11 is steadily increasing its market share among Microsoft operating systems [76], so testing on it could be beneficial.

References

[1] B. LaPorte. (2024) "History of Ransomware: The Evolution of Attacks and Defense Mechanisms". [Accessed: 02-09-2024]. [Online]. Available: https://blog.morphisec.com/ransomware-history-evolution-of-attacks-and-defenses

[2] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–37, Jan. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3514229

[3] Statista. (2024) "Ransomware - Statistics & Facts". [Accessed: 11-09-2024]. [Online]. Available: https://www.statista.com/topics/4136/ransomware/#topicOverview

[4] R. Moussaileb, N. Cuppens, J.-L. Lanet, and H. L. Bouder, "A Survey on Windows-based Ransomware Taxonomy and Detection Mechanisms," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, Jul. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3453153

[5] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A Real-Time Detection System Against Cryptographic Ransomware," in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, vol. 11050, pp. 114–136, series Title: Lecture Notes in Computer Science.

[6] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. Nara, Japan: IEEE, Jun. 2016, pp. 303–312. [Online]. Available: http://ieeexplore.ieee.org/document/7536529/

[7] P. M. Rebecca Bace, "Intrusion Detection Systems."

[8] S. Samonas and D. Coss, "The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security."

[9] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware."

[10] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "ShieldFS: a self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. Los Angeles California USA: ACM, Dec. 2016, pp. 336–347. [Online]. Available: https://dl.acm.org/doi/10.1145/2991079.2991110

[11] A. Kharraz and E. Kirda, "Redemption: Real-Time Protection Against Ransomware at End-Hosts," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds. Cham: Springer International Publishing, 2017, vol. 10453, pp. 98–119, series Title: Lecture Notes in Computer Science.

[12] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, Jan. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320311002901

[13] C. Beaman, A. Barkworth, T. D. Akande, S. Hakak, and M. K. Khan, "Ransomware: Recent advances, analysis, challenges and future research directions," *Computers & Security*, vol. 111, p. 102490, Dec. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016740482100314X

[14] Cloudfare. "What was the WannaCry ransomware attack?". [Accessed: 02-09-2024]. [Online]. Available: https://www.cloudflare.com/learning/security/ransomware/wannacry-ransomware/

[15] "Wikipedia". (2024) "Uniform distributon". [Accessed: 02-09-2024]. [Online]. Available: https://www.statista.com/statistics/1410498/ransomware-revenue-annual/#:~:text=Total%20value%20received%20by%20ransomware%20attackers%20worldwide%202017%2D2023&text=In%202023%2C%20the%20total%20amount,dollars%20in%20the%20year%20prior

[16] Clusit. (2024) "Rapporto Clusit 2024 sulla sicurezza ICT in Italia". [Accessed: 02-09-2024]. [Online]. Available: https://www.opificiumagazine.it/wp-content/uploads/2024/03/Rapporto-Clusit-2024.pdf

[17] H. L. Kevin Savage, Peter Coogan. (2015) "Security Response: The Evolution ransomware". [Accessed: 02-09-2024]. [Online]. Available: https://its.fsu.edu/sites/g/files/imported/storage/images/information-security-and-privacy-office/the-evolution-of-ransomware.pdf

[18] T. M. Research, "Rethinking Tactics: 2022 Annual Cybersecurity Report," 2022.

[19] W. Alraddadi and H. Sarvotham, "A Comprehensive Analysis of WannaCry: Technical Analysis, Reverse Engineering, and Motivation."

[20] R. E. Alex Berry, Josh Homan. (2017) "WannaCry Malware Profile". [Accessed: 02-09-2024]. [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/wannacry-malware-profile/

[21] D. Murphy. (2024) "Common Ransomware Encryption Techniques". [Accessed: 02-09-2024]. [Online]. Available: https://www.lepide.com/blog/common-ransomware-encryption-techniques/

[22] B. Enterprise. (2022) "What Are Ransomware Families? (And Why Knowing Them Can Help Your Business Avoid Attack". [Accessed: 02-09-2024]. [Online]. Available: https://www.bitdefender.com/blog/businessinsights/what-are-ransomware-families-and-why-knowing-them-can-help-your-business-avoid-attack/?srsltid=AfmBOoo82XbpbS0SXNzpKXcT-7rIa5XpdPQ03eUlY-hCRf5T4OJyqz-i%2F

[23] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, Jan. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804512001944

[24] M. De Vivo, E. Carrasco, G. Isern, and G. O. De Vivo, "A review of port scanning techniques," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 41–48, Apr. 1999. [Online]. Available: https://dl.acm.org/doi/10.1145/505733.505737

[25] M. Team. (2018) "OS Detection with nmap". [Accessed: 02-09-2024]. [Online]. Available: https://nmap.org/book/man-os-detection.html

[26] S. Chakrabarti, M. Chakraborty, and I. Mukhopadhyay, "Study of snort-based IDS," in *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*. Mumbai Maharashtra India: ACM, Feb. 2010, pp. 43–47. [Online]. Available: https://dl.acm.org/doi/10.1145/1741906.1741914

[27] O. Al-Jarrah and A. Arafat, "Network Intrusion Detection System using attack behavior classification," in *2014 5th International Conference on Information and Communication Systems (ICICS)*, Apr. 2014, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6841978

[28] K. M. Elleithy, D. Blagovic, W. K. Cheng, and P. Sideleau, "Denial of Service Attack Techniques: Analysis, Implementation and Comparison," vol. 3, no. 1.

[29] W. Pires, T. De Paula Figueiredo, Hao Chi Wong, and A. Loureiro, "Malicious node detection in wireless sensor networks," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Santa Fe, NM, USA: IEEE, 2004, pp. 24–30. [Online]. Available: http://ieeexplore.ieee.org/document/1302934/

[30] D. Gumusbas, T. Yldrm, A. Genovese, and F. Scotti, "A Comprehensive Survey of Databases and Deep Learning Methods for Cybersecurity and Intrusion Detection Systems," *IEEE Systems Journal*, vol. 15, no. 2, pp. 1717–1731, Jun. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9099844/

[31] Song Han, Miao Xie, Hsiao-Hwa Chen, and Yun Ling, "Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges," *IEEE Systems Journal*, vol. 8, no. 4, pp. 1052–1062, Dec. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6942184/

[32] Wikipedia. (2024) "Machine learning features". [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Feature_(machine_learning)

[33] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, 1st ed. Cambridge University Press, May 2014. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781107298019/type/book

[34] F. Jáñez-Martino, R. Alaiz-Rodríguez, V. González-Castro, E. Fidalgo, and E. Alegre, "A review of spam email detection: analysis of spammer strategies and the dataset

shift problem," *Artificial Intelligence Review*, vol. 56, no. 2, pp. 1145–1173, Feb. 2023. [Online]. Available: https://link.springer.com/10.1007/s10462-022-10195-4

[35] Wikipedia. (2024) "Regularization". [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Regularization_(mathematics)

[36] R. Ali, A. Ali, F. Iqbal, M. Hussain, and F. Ullah, "Deep Learning Methods for Malware and Intrusion Detection: A Systematic Literature Review," *Security and Communication Networks*, vol. 2022, pp. 1–31, Oct. 2022. [Online]. Available: https://www.hindawi.com/journals/scn/2022/2959222/

[37] A. S. Dina and D. Manivannan, "Intrusion detection based on Machine Learning techniques in computer networks," *Internet of Things*, vol. 16, p. 100462, Dec. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660521001037

[38] R. Rijtano. (2024) "I dati sanitari di migliaia di italiani sono ormai online". [Accessed: 02-09-2024]. [Online]. Available: https://www.guerredirete.it/i-dati-sanitari-di-migliaia-di-italiani-sono-ormai-online/

[39] M. Cen, F. Jiang, X. Qin, Q. Jiang, and R. Doss, "Ransomware early detection: A survey," *Computer Networks*, vol. 239, p. 110138, Feb. 2024. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128623005832

[40] U. Urooj, B. A. S. Al-rimy, A. Zainal, F. A. Ghaleb, and M. A. Rassam, "Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions," *Applied Sciences*, vol. 12, no. 1, p. 172, Dec. 2021. [Online]. Available: https://www.mdpi.com/2076-3417/12/1/172

[41] D. Hitaj, G. Pagnotta, F. De Gaspari, L. De Carli, and L. V. Mancini, "Minerva: A File-Based Ransomware Detector," Apr. 2024, arXiv:2301.11050 [cs]. [Online]. Available: http://arxiv.org/abs/2301.11050

[42] D. Sanvito, G. Siracusano, R. Gonzalez, and R. Bifulco, "MUSTARD: Adaptive Behavioral Analysisfor Ransomware Detection."

[43] C. Q. Vassil Roussev. (2013) "Similarity". [Accessed: 02-09-2024]. [Online]. Available: http://roussev.net/sdhash/tutorial/03-quick.html#digest-generation

[44] ”Wikipedia”. (2024) ”Entropy”. [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(information_theory)

[45] ——. (2024) ”Uniform distributon”. [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Discrete_uniform_distribution

[46] ——. (2024) ”Entropy max value”. [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(information_theory)#Example_2

[47] Cuckoo. (2019) ”Cuckoo sandabox”. [Accessed: 02-09-2024]. [Online]. Available: https://cuckoosandbox.org/

[48] Wikipedia. (2018) ”Sandbox”. [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Sandbox_(computer_security)

[49] Virustotal. (2024) ”Virustotal”. [Accessed: 02-09-2024]. [Online]. Available: https://www.virustotal.com/gui/home/upload

[50] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, and O. S. Pianykh, “Temporal quality degradation in AI models,” *Scientific Reports*, vol. 12, no. 1, p. 11654, Jul. 2022. [Online]. Available: https://www.nature.com/articles/s41598-022-15245-z

[51] M. Kull and P. Flach, “Patterns of dataset shift.”

[52] M. Oleszak”. (2024) ”Detecting Concept Shift: Impact on Machine Learning Performance”. [Accessed:06-09-2024]. [Online]. Available: https://towardsdatascience.com/detecting-concept-shift-impact-on-machine-learning-performance-16923261cda8

[53] D. Ruano-Ordás, F. Fdez-Riverola, and J. R. Méndez, “Concept drift in e-mail datasets: An empirical study with practical implications,” *Information Sciences*, vol. 428, pp. 120–135, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025516320655

[54] W. A. Knaus, “Prognostic Modeling and Major Dataset Shifts During the COVID-19 Pandemic: What Have We Learned for the Next Pandemic,” *JAMA Health Forum*, vol. 3, no. 5, p. e221103, May 2022. [Online]. Available: https://doi.org/10.1001/jamahealthforum.2022.1103

[55] M. Bazaar". (2024) "MalwareBazaar". [Accessed: 06-09-2024]. [Online]. Available: https://bazaar.abuse.ch/

[56] "Microsoft". (2023) "IRPs Are Different From Fast I/O". [Accessed: 08-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/irps-are-different-from-fast-i-o

[57] ——. (2023) "Windows Kernel-Mode I/O Manager". [Accessed: 02-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

[58] ——. (2023) "IRP structure". [Accessed: 08-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_irp

[59] MalwareBazaar. (2024) "MalwareBazaar API". [Accessed: 06-09-2024]. [Online]. Available: https://bazaar.abuse.ch/api/

[60] O. Uchubilo. (2023) "Detecting BlackCat ransomware with Wazuh". [Accessed: 09-09-2024]. [Online]. Available: https://wazuh.com/blog/detecting-blackcat-ransomware-with-wazuh/

[61] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. S2–S11, Sep. 2009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1742287609000346

[62] N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Generating realistic *impressions* for file-system benchmarking," *ACM Transactions on Storage*, vol. 5, no. 4, pp. 1–30, Dec. 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1629080.1629086

[63] REMnux. (2020) "REMnux: A Linux Toolkit for Malware Analysis". [Accessed: 02-09-2024]. [Online]. Available: https://remnux.org/

[64] "Microsoft". (2023) "IRP struct". [Accessed: 04-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fltkernel/ns-fltkernel-_flt_parameters

[65] ——. (2023) ”IRP Major Function Code”. [Accessed: 04-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-major-function-codes

[66] ——. (2023) ”FLT_PARAMETERS union”. [Accessed: 04-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fltkernel/ns-fltkernel-_flt_parameters

[67] ——. (2024) ”File Information Class”. [Accessed: 04-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ne-wdm-_file_information_class

[68] ——. (2023) ”RP_MJ_SET_INFORMATION”. [Accessed: 04-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/irp-mj-set-information

[69] ”Wikipedia”. (2024) ”I/O operation”. [Accessed:04-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Input/output

[70] ”Microsoft”. (2024) ”Driver Signing”. [Accessed: 02-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing

[71] ——. (2021) ”Digital Signatures”. [Accessed: 02-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/digital-signatures

[72] ——. (2023) ”Windows API 32”. [Accessed: 03-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

[73] ——. (2023) ”Open a file in Wind”. [Accessed: 03-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfileai

[74] ”Wikipedia”. (2024) ”User space and kernel space”. [Accessed: 02-09-2024]. [Online]. Available: https://en.wikipedia.org/wiki/User_space_and_kernel_space

[75] ”Microsoft”. (2021) ”Windows Kernel-Mode I/O Manager”. [Accessed: 02-09-2024]. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-i-o-manager

[76] Statista. (2024) ”Windows OS market share by version”. [Accessed: 11-09-2024]. [Online]. Available: https://www.statista.com/statistics/993868/worldwide-windows-operating-system-market-share/

Acknowledgments

Un grazie a tutti quelli che mi hanno supportato e sopportato in questo percorso.