An Anomaly Detection System For Early Detection of Ransomware Attacks

Candidate Number: 1069231

Department of Computer Science

University of Oxford

A Dissertation submitted for the degree of

Master of Science in Advanced Computer Science

Trinity 2023

(Page left intentionally blank)

Abstract

Ransomware is a type of malware that infects a device and spreads rapidly to encrypt files or to lock the device, restricting the users from accessing them. The attackers then demand payment, in exchange for providing the decryption key or unlocking the device. Additionally, they may steal critical files from the victim's machine and threaten to publish them online in case the recovery of the system and the files is possible. Lately ransomware have grown to become a serious threat to many organizations and individuals, resulting in high financial, legal, and sometimes social costs. However, the current approaches for detecting malware and specifically ransomware are not effective, especially since ransomware must be detected as early as possible. In this dissertation I propose a method to detect ransomware attacks using a combination of features extracted from the phases before, during and after encryption has occurred, by training a machine learning-based anomaly detection model. To train the model I monitored a test system for several days, to establish a baseline of normal system behavior and I deployed a self-developed ransomware along with the model on the system. The results show that the model can detect ransomware activity before encryption has occurred, resulting in the early detection of ransomware, while also generating no false positive alerts from normal behaviors.

Acknowledgments

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Since the creation of the modern computer, adversaries have crafted ways to leverage its capabilities for their own personal, financial, and political interests. Malicious practices and software, also known as malware, have been used to harm many individuals and organizations. Over the latest decades and especially after the COVID-19 pandemic, a special type of malware called ransomware has quickly become one of the most critical threats to security, impacting individuals and enterprises, the consequences of which range from minor information and financial losses to important societal aftereffects.

Ransomware is a type of malware that infects a system and limits access to important user files and assets until a ransom, or payment is made to the attackers. In modern ransomware, attackers additionally exfiltrate vital data to a remote server they control, in a double extortion attack, to put more pressure on the victim. Ransomware typically achieve that by either swiftly encrypting files, which contain sensitive information or are important for the normal operation of the victim's activities, or by generally locking the whole device and rendering it unusable.

While most victims refuse to pay the ransom, due to the high volume and destructiveness of ransomware attacks, ransomware have become a billion-dollar criminal industry and attackers are continuously devising new ways to extort more money. In fact, it is predicted by Cybersecurity Ventures that ransomware will cost its victims approximately $265 billion (USD) annually by 2031 https://conceal.io/2023-whos-who-in-ransomware-report/. But while in the past the usual costs incurred by ransomware infections were mostly financial, ransomware have also started attacking critical infrastructure such as the healthcare industry. Most notably, the infamous WannaCry ransomware attacks crippled hundreds of organizations many of which were hospitals. Additionally, during the COVID-19 pandemic many other healthcare organizations and hospitals were compromised as the need to operate without disruptions made these organizations an interesting target.

To mitigate the risks of ransomware attacks, individuals and organizations are advised to take precautionary and proactive measures, such as employing up-to-date intrusion detection systems, robust monitoring and incident response tools, regular backups, undergoing awareness and security training and avoiding paying the ransom in the case of an infection. Additionally, both industry and research have started to primarily focus their attention on the early detection of ransomware attacks, discovering ongoing attacks before the ransomware has encrypted the assets or has exfiltrated the target files.

Nevertheless, massive ransomware campaigns continue to take place, with new families and variants specially crafted to evade the defense mechanisms in place. This can only mean that the current systems employed lack the novel early detection methods required to stop ransomware before they achieve their objectives.

1.1 MOTIVATION

While significant progress has been made, traditional security measures have struggled to keep pace with the rapidly evolving ransomware landscape. Signature-based anti-virus solutions and conventional intrusion detection systems often rely on known malware signatures, rendering them ineffective against novel and unseen ransomware variants. Additionally, attackers often deploy techniques to conceal their activities and evade detection systems, allowing them to conduct attacks without raising alarms.

To defend against ransomware, the proactive and timely detection of attacks is vital. Anomaly detection systems, such as Auto-Encoders have shown their promise in many fields including cyber security doi: 10.3390/s21134294 and can bridge the gap left by traditional security solutions. By learning the normal behavior patterns of processes, these systems can recognize ongoing ransomware attacks, as their behavior deviates significantly from legitimate behavioral patterns and can be employed as a part of an organization's holistic defense infrastructure. The fundamental idea of Anomaly Detection systems is to establish a baseline of normal behavior by exposing them to legitimately operated environments and then employ advanced learning algorithms to detect any deviations that may indicate the presence of ransomware in the system.

The current research on ransomware detection unfortunately has plenty of important limitations. Frequently, works focus on the general classification of ransomware, aiming to generally identify ransomware based on their static characteristics (file signatures, PE structure, strings etc) and some extracted behavioral patterns, but do not consider the timing of the detection. Even though static analysis of malware can detect threats before they are executed, polymorphic and metamorphic ransomware, that is ransomware that can change their identifiable characteristics, are efficient in evading such defense measures.

Other works, while they provide early detection examples, tend to generate a significant amount of false positive alerts, which can inhibit the detection and response efforts of security systems and teams. After analysis, one reason for the generation of false positive alerts is usually the lack of diversity in their feature set. Also, detection systems that are created around features that are easy to be bypassed by attackers have profound drawbacks. Strong detection systems should rely on a diverse feature set, capable of capturing attacks in their nascent stages while simultaneously encompassing attempts to evade defense mechanisms.

Undoubtedly, many studies show auspicious results in the classification and early detection fields and the purpose of this work is to combine these promising methods and produce a novel and robust detection system.

All the above show that the defense mechanisms usually employed could and should be enriched with a novel detection system that is influenced by the strengths and limitations of previous works and leverages the power of anomaly detection systems. An anomaly detection system, with its capacity to identify ransomware behavior at their early stages, holds the promise of quick intervention, potentially mitigating the extent of damage and enabling timely responses to thwart the attack.

1.2 CONTRIBUTIONS

This dissertation aims to make significant contributions to the field of cyber security by proposing and creating an anomaly detection system designed specifically for early detection of ransomware attacks focusing on the most common ransomware encountered, crypto-ransomware. It endeavors to analyze the shortcomings of previous works and traditional security measures and offer a novel approach to enhance the security of systems against ransomware attacks.

Utilizing previous works that analyze ransomware behaviors, this paper investigates a variety of features that correspond to the various stages of a ransomware infection. That feature set is intended to extract attributes from the general behavior of processes, their network traffic, file access patterns and registry interactions and create a comprehensive dataset capable of describing and encapsulating ransomware activity and most notable its contrast to legitimate behavior. Supplementarily it endeavors to form a basis for further exploration of features on identifying ransomware behaviors.

Using that diverse feature set the proposed system leverages machine learning-based anomaly detection techniques to establish a normal baseline and detect deviations that are indicative of ransomware infections. Specifically, the capabilities of an Auto-Encoder are explored as this type of model, despite its simplicity has shown the strong capabilities in detecting anomalous behaviors.

To evaluate the performance of the proposed anomaly detection system, the system is tested in a legitimately operated system along with a self-developed ransomware that performs the most common tasks of ransomware met in the wild. Performance metrics such as false positive rate, false negative rate, and the stage at which the attack is detected are meticulously analyzed to assess and improve the effectiveness of the system in early ransomware detection.

Furthermore, this dissertation aims to contribute to the broad cyber security research community as well. It seeks to emphasize the importance of novel early detection systems, promote other works that focus on ransomware attacks, provide a basis for a robust feature set that can be expanded to include more attributes that help detect ransomware and finally, to showcase the utility of machine learning-based anomaly detection systems in the battle against ransomware.

In conclusion, this dissertation aims to advance the area of early detection of ransomware attacks, provide the basis for and encourage further research and creation of systems, and overall assist individuals and organizations to defend themselves against the potent threat of ransomware.

1.3 STRUCTURE OF THE DISSERTATION

2 Background

In this chapter I first describe the nature and objectives of ransomware and the way they operate to conclude an attack. Subsequently, I present common approaches to detecting ransomware attacks and finally I describe Anomaly Detection Systems and Auto-Encoders. After reading this chapter, the reader will be able to grasp what are ransomware attacks, usual techniques employed to detect ransomware and Auto-Encoder models for anomaly detection.

2.1 Ransomware

As mentioned before, ransomware is a class of malicious software, or malware that threatens to permanently block users from accessing their files or systems, until a ransom is paid to the attackers. In a "double extortion" attack, the attackers may additionally threaten to publish the victim's data on the Internet unless that ransom is paid. Ransomware do that by firstly infecting a target machine, rendering the entire machine or a set of files inaccessible to the user and demanding a payment to recover that access. Additionally, modern ransomware steal files that the attackers may find useful to servers that they control and threaten the victims to publish them. Since the assets that are compromised contain vital information for the users and may be crucial for the normal activities of the victims, their inaccessibility puts pressure on them to pay the ransom to the attackers, regain access to them, and continue their operations.

Overall, ransomware can be split into two broad categories based on the way they operate: Crypto-Ransomware or Encryptors and Locker Ransomware.

Crypto-Ransomware or *Encryptors* are the most common types of ransomware and are designed to encrypt data, information and files on the victim's device. Although the victims would be able to see the encrypted assets and use the device, they would not be able to access the data since the data are encrypted. After the encryption, the ransomware notifies the users that their data have been encrypted, usually by placing a note in specific directories and prompts them to make a payment until a specific deadline to retrieve them.

Locker-Ransomware work similarly to encryptors, but with the difference that they do not encrypt, destroy, or alter the users' files. Instead, they work by locking the users completely out of their systems. Usually, the victim is allowed to interact with the system minimally; he can view only the lock screen or interact only with the screen displaying the ransom demand. Similarly, to encryptors, a deadline may be given to the victims to pressure them into paying the ransom.

Ransomware have generally been around since 1989, when the PC Cyborg attack, the first incident classified as a ransomware, was documented. With the passage of time and as the internet boomed, ransomware attacks have evolved, and many families and strains have been created incorporating complex mechanisms to achieve their goals and evade the defenses of the victims.

Early ransomware were mostly encryptors targeting predominantly Windows machines. Additionally, early ransomware were using exclusively symmetric ciphers to encrypt the target's files. GPCode was infecting machines via phishing emails and utilized custom symmetric encryption algorithms. Later, Archiveus, Krotten and more started utilizing asymmetric encryption algorithms such as RSA, to guarantee that if somehow the key used for file encryption was discovered, there would be no way to decrypt the files except by obtaining the private key residing at the attacker's machines.

A few years later, WinLock heralded the rise of locker-ransomware. Instead of encrypting the victim's files, WinLock would lock the user out of their desktop and demand payment via a paid SMS.

Nowadays, ransomware utilize more sophisticated encryption and extortion methods, employing *hybrid encryption* and *double extortion*. Hybrid encryption entails the use of symmetric keys for the encryption of the target files and the encryption of the keys themselves using asymmetric keys. In this way, the attackers can leverage the speed of symmetric encryption for the encryption of the files and the confidentiality offered by asymmetric ciphers to conceal the symmetric key. In a double extortion attack, attackers do not only encrypt the victim's files, but they also steal them and threaten to publish them online, unless the requested ransom is paid.

The most alarming evolution of ransomware that dominates today's threat landscape is the emergence of *Ransomware-as-a-Service*. RaaS is a business model between ransomware developers/operators and affiliates in which the latter rent the services of the former in underground markets and pay to launch a ransomware attack created and managed by the operators. RaaS has skyrocketed ransomware attacks, especially because it is not anymore necessary for the attackers (affiliates) to possess neither sophisticated offensive capabilities nor time to successfully conduct a ransomware attack. On the contrary RaaS kits are aimed to be user-friendly, offer user reviews, can track the status of their attack, and even offer 24/7 support. This way, anyone with enough malicious intent and limited skills can conduct a successful attack by leveraging the services offered by potent ransomware authors.

To illustrate, DarkSide is a RaaS group that is believed to be behind the Colonial Pipeline ransomware attack, an attack that incapacitated for several days large and vital pipelines in the United States. The threat actors, instead of developing their own attacks, leveraged the RaaS offered by DarkSide and performed one of the largest publicly disclosed cyber-attacks against critical infrastructure in the US.

2.1.1 Targets of Ransomware

Ransomware in general attack a variety of devices, operating systems, and groups of people. Usually, they are designed for a particular platform and operating system because they often take advantage of the specific capabilities offered by that type of target, such as libraries, functions, architecture, or vulnerabilities.

That being said, ransomware are observed to target PCs and workstations running Windows (WannaCry), macOS (Patcher) and GNU/Linux (RansomEXX). Additionally, they can attack mobile devices, but mostly Android devices, as Apple has a strict and heavily controlled application ecosystem, hardening the entry of ransomware applications to iOS devices. Finally, IoT and Cyber-Physical Systems (CPSs) can also be affected by ransomware, although they are not the primary focus of attackers. As though these devices are becoming more ubiquitous in domestic and industrial environments, attackers are expected to shift their focus to them, and exploit the poor defense mechanisms employed in these not so well protected devices.

Although the targets can be diverse, the most affected type and operating system are PCs/workstations running Microsoft Windows. As a result of its high popularity among users, attackers favor it and have focused their attention on creating ransomware specially crafted to attack Windows machines https://doi.org/10.1145/3514229.

Eventually, because crypto-ransomware are the most common threat, in this work I focus on the study and detection only of encryption-based attacks, noting that many of the attack steps exhibited by crypto-ransomware are observed also in locker-ransomware as well. Additionally, I center my attention around the Microsoft Windows operating system as it is the most common ransomware target. Therefore, for the rest of this work any mentions to ransomware are supposed to be crypto-ransomware that target Windows.

2.1.2 The Nature of Ransomware; The Ransomware Kill Chain

Ransomware, as any kind of malware must take certain steps to operate. They typically spread to the victims via phishing emails, malicious downloads, or by exploiting vulnerabilities in the system and deceiving the victim into downloading them and running the malicious payload. To fully comprehend the way ransomware operate it is useful to group and map their actions to a framework similar to the one presented by Stamper, which resembles the Cyber Kill Chain.

Reconnaissance

First and foremost, the attackers conduct reconnaissance activities to gather intelligence about the victims, as ransomware are usually targeted. Although some malware exhibit worm behavior to propagate, they usually propagate to systems that satisfy a certain number of criteria that match the desired victims. For example, they may have a list of IP addresses to or not to attack, or systems that reside into specific regions as was done by Sodinokibi, a ransomware that avoided attacking computers from countries that were formerly part of the USSR https://www.acronis.com/en-gb/blog/posts/sodinokibi-ransomware/, by checking the locale settings. In this stage, ransomware may also conduct network scans and view the running processes to uncover any useful information.

Weaponization

When a target is identified, the attackers craft the attack vectors that will be used to conduct the attack. At this point, they craft specialized emails that will be sent to the targets and the seemingly benign, but in reality, malicious links and attachments that will be contained in the emails. They can also upload their malicious files to websites, so the users can download them following a supply chain compromise or phishing attack.

Delivery

Ransomware are typically delivered to the victims via phishing emails, when a user clicks on the crafted malicious links or attachments, or when a user is redirected to websites containing the malicious files. For example, Ryuk used phishing campaigns which redirect unaware users to documents hosted in legitimate hosting services, such as Google Drive. Furthermore, the delivery can be conducted by Supply Chain attacks, where a trusted third party is compromised and used to distribute malware or may be dropped by other cooperating malware that previously infected the victim.

Exploitation and Privilege Escalation

Many of the actions that ransomware need to take (deletion of backups, halting of processes and utilities, interaction with the Windows registry, access to specific directories and files) require elevated privileges. To escalate their privileges, ransomware may exploit vulnerabilities, may use code injection techniques to inject malicious code into normal elevated processes, or if exploitation is unsuccessful, they may directly request from the user to give elevated privileges to the process using Windows User Account Control prompts. While vulnerability exploitation is the most preferred way to gain elevated privileges, when it is not possible, ransomware such as Sodin, use UAC prompts.

Installation

Ensuring that the malware has sufficient privileges to perform its actions, it unpacks and decrypts its payload so that its files are accessible to be used for the attack. Next, the ransomware check if the machine is included in a list that contains devices that should not be attacked, as explained in the Reconnaissance phase. If the machine is included in the whitelist, the attack is aborted.

Subsequently, ransomware tend to terminate processes that may detect malicious activity, inhibit file encryption and file access, or help recovery of the system. Antivirus software, and Database Management Systems (DBMSs) such as MySQL are terminated, as they might detect the ransomware or prevent it from accessing and encrypting files on the device.

As attackers want to be able to gain a strong foothold into the device and remain there, ransomware try to gain *persistence*. To do that, it is common that they modify the Windows Registry to add themselves as an AutoRun program. For example, this behavior is exhibited by Ryuk and Wannacry.

Command and Control

As with many other malware families, ransomware may utilize Command and Control (C&C) servers to gather information, coordinate the attack, update the malware and exfiltrate data in case of a double extortion attack. At this stage, the victim establishes a connection with the C&C server, which may be done via custom protocols, the TOR network, or may blend with legitimate traffic, for example by using HTTPS to transmit all the necessary information in an encrypted fashion. In the early stages of the attack, the ransomware may send to the C&C important information about the victim's device, such as OS, Machine Name, basic Network topology and other information that will help the proper configuration and success of the attack and also the propagation of the malware to other systems. Moreover, the C&C server may transmit the encryption keys that will be used for the encryption of the files, or the encryption of the key that will be used to encrypt the files, if hybrid encryption is used.

Self-Propagation

While self-propagation is not a necessary step to conclude the attack, ransomware that want to maximize the number of victims may be designed to spread automatically to other machines. It is common that ransomware propagate within the LAN after the infected machine performs network scans to identify potentially vulnerable hosts. For example, Ryuk reads the victim's ARP cache and sends Wake-on-Lan messages to each entry, so that when they are online it can send an ICMP ping to all addresses in a range of subnets. After identifying potential victims, the ransomware attempts to spread to those machines using a variety of methods such as mounting shared resources and copying itself to the new victim.

Actions on Objectives

At this phase, the stage is almost set for the attack to perform its main function; encrypt and potentially exfiltrate the data. The few parts remaining relate to the *discovery* of the directories and files that the ransomware wants to encrypt. The ransomware iterates through all directories in the machine to discover target files and some also access network attached drives on the LAN.

It is important to note that ransomware do not blindly encrypt all the files they can find in each directory. They usually target specific file extensions or directories that contain critical data and can disrupt the normal operation of the organization https://arxiv.org/abs/1609.03020. Ransomware usually target file types that may be used to store information that are critical for the users or for the organization such as documents that may contain sensitive personal and financial information, databases, configuration files and other assets that are crucial for the operation of business processes. On the other side, ransomware may avoid certain directories and files that may inhibit the rest of the attack or are easy to recover. For that reason, critical Windows files, web browsers and executables, are not targeted as they are used by the user to minimally interact with the device, to get notified about the infection, to pay the ransom or can be easily and quickly recovered.

Once the desired files have been located, ransomware utilizing double extortion exfiltrate the data to the C&C server. Ransomware have been observed to exfiltrate data at various points in the kill chain, for example before or after encryption. Nevertheless, exfiltration occurs after the directories are iterated and the target files have been located. For the data exfiltration attackers may use existing C&C channels, over network mediums such as Wi-Fi connections, Bluetooth, or other radio frequency channels, over Web Services and Cloud accounts. For example, they may use Google Drive, or Cloudflare to blend their activity with the legitimate activity of hosts.

Reaching this stage, ransomware are ready to encrypt the discovered files. Although there exist many methods to encrypt the files, modern ransomware use hybrid encryption where the file is encrypted using a symmetric cipher, such as AES, and then the symmetric key itself is encrypted using an asymmetric public key, saved, and potentially transmitted to the C&C, who owns the private key. That asymmetric key, and perhaps the symmetric key itself, may have been transmitted by the C&C server at an earlier stage or they may have been hardcoded in the ransomware itself. It is known that symmetric ciphers are faster than asymmetric ones, so as attacks at this stage depend on their speed, attackers prefer to speed up this process as much as possible, by using symmetric ciphers and multithreading for file encryption.

To show that a file is encrypted, attackers may choose to rename the encrypted files by appending an extension. For instance, Babuk adds the "*.babyK"* extension and Wannacry appends the "*.WNCRY"*.

In addition to that, ransomware typically delete all backups, by scanning for directories containing the word 'Backup' or similar, and interacting with known backup utilities such as Windows Shadow copies. Additionally, recovery tools are disabled, such as the Windows Recovery Environment, so that recovery of the system is impossible.

Extortion

To conclude the attack, the attacker notifies the users that their files are encrypted by placing a ransom note in each affected directory and awaits payment. That ransom note explains to the victims that their files are encrypted, demands a ransom payment for their recovery and for them not to be published, assures them that there is no way of recovering the files unless the ransom is paid and provides information related to the payment of the ransom.

Of course, this abstract kill chain cannot encapsulate and accurately describe all the ransomware that are met in the wild and their nuances, but succeeds in providing sufficient and detailed description and enumeration of the most common steps observed in ransomware and sometimes the order that they happen in. Therefore, it is a very valuable tool when analyzing ransomware behavior and especially when building defense mechanisms to detect ransomware, as this work aims to do. Throughout the rest of this work, I use this kill chain to understand what features can be constructed to detect ransomware activity and what activity they aim to catch.

2.2 ANALYZING AND DETECTING RANSOMWARE

This section focuses on the analysis of the common methods that research and industry products employ when they analyze and detect ransomware attacks. Analysis precedes detection, as detection systems use the findings of their study to infer if a certain observation is a ransomware.

There are commonly two major types of analysis that can be performed and that is used for the detection of ransomware. Although there are many ways that they can be categorized, here I present a taxonomy based on the way that the findings are extracted. Additionally, these methods are not specific to ransomware detection but are used generally during malware analysis.

2.2.1 Static Analysis and Signature-Based Detection

Static analysis strives to examine a sample and infer if it is a ransomware by extracting information without running it. This is typically done by disassembling and analyzing the code of binaries to extract information regarding their structure, contents, and other static characteristics and identify any intent for malicious activity.

The data that can usually be extracted are file hashes, header information, strings, opcodes and file types. Researchers obtain these features from samples without running them and can use them to build a profile of the sample, also called a signature. Signature-based detection is based on the discovered signatures of known ransomware and during detection such systems try to match analyzed samples with a knowledge base of previously discovered malicious signatures. In the case of a match, they infer that this sample has malicious intent, due to its observation in the past, and raises an alert.

Static analysis is fast and it is performed without the risk of infecting the analysis environment, as the sample is not executed. Therefore, a successful static analysis can add to the early detection of ransomware and detect them before even they are executed. However, signature-based detection has important limitations. Firstly, it is inefficient against novel ransomware not seen again in the past, as their signatures do not exist in their signature databases. Malware authors employ concealment techniques, such as obfuscation, polymorphism, metamorphism, and encryption to either conceal or constantly change their static characteristics. This way, they can make static analysis efforts harder, and evade defense systems that rely on features obtained during static analysis.

2.2.2 Dynamic Analysis and Behavior-Driven Detection

Dynamic analysis on the other hand does not rely on the static properties of the samples under examination, but instead requires running the sample and observe its behavior to determine if it is a ransomware or not. During research, dynamic analysis of the sample is usually done by running the samples inside an isolated environment (a sandbox), to analyze the sample without risking damage to the device. However, in real-time detection systems, this is not available as the number of samples that should be analyzed is large. In such cases, dynamic analysis relies on the real-time extraction of behavioral features, by constantly monitoring the system. Of course, dynamic analysis-based detection systems use features that were discovered during sandbox analysis that they were important for ransomware detection, but do not rely on sandbox environments for real-time detection.

Dynamic analysis aims to obtain behavioral features such as logs, process activity, file system activity, I/O access patterns, function/API/system calls, network activity, registry activity, resource usage and sensor readings. As will be illustrated in Chapter 3, not all these features have significant strengths in detecting ransomware attacks and not one of the features can solely detect attacks. Instead, I advocate that a robust system should combine features and techniques that have shown their promise and resilience in distinguishing ransomware attacks from normal behavior and detecting them in their earliest stages.

Since dynamic analysis relies on the observation of its behavior during execution, it is costly in terms of time and resources compared to static analysis and risks the damage of the device if the detection mechanism employed is not efficient enough. Nevertheless, concealment techniques that are employed by ransomware to evade static analysis-based detection are not effective against dynamic analysis as they cannot conceal the behavior of the ransomware. On the other hand, ransomware authors are changing the behavior of their malware, to mislead and avoid even dynamic analysis systems by making their activities seem normal. For this reason, a robust system of a diverse and granular feature set is imperative to effectively capture the various methods that ransomware try to masquerade their activity as normal behavior.

There is a myriad of methods that are used to build detection systems using behavioral features, such as statistics-based, deception-based and machine-learning based. Statistics-based detection relies on the capture of the statistical properties of the behavioral features extracted and uses tests such as the chi-square goodness-of-fit test to detect attacks. Deception-based methods take a different approach. Instead of examining the statistical properties of data, they deploy several decoy files around the environment, that may seem interesting targets to attackers and monitor any interactions with these files. Assuming that these decoy files should not be interacted with during normal operations, deception-based systems assume that any activity on those files indicates suspicious activity, and thus an alarm should be raised.

Machine-learning based detection seems to dominate the behavior-based detection systems. They leverage the power and speed of machine learning algorithms to automatically learn the patterns and intricacies of the data and detect the behavioral patterns of ransomware attacks. Machine learning techniques in this context can broadly split into two other main categories, classifiers, and anomaly detection methods. Classifiers, rely on the existence of sufficient examples of malicious examples to learn from, but malicious samples are generally very scarce, rendering these methods hard to use. On the other hand, anomaly detection methods rely on the existence usually only of normal activity in the data used. They work by learning the trends and characteristics solely of normal behaviors, leveraging their enormous supply of examples and they detect attacks by behaviors that deviate greatly from that established baseline of normal.

In this work, I focus on the extraction of behavioral features of ransomware and normal operations that can accurately detect ransomware attacks. I employed a machine learning-based anomaly detection method leveraging the abundance of normal behavior activities and avoiding the problem of scarcity of ransomware examples. In the following section I delve deeper into the nature of anomaly detection methods, and the type of technique that I used, the Auto-Encoders.

2.3 Anomaly Detection

Anomaly Detection, also called Outlier Detection or Novelty Detection refers to the task of identifying data points or patterns that do not conform to expected behaviors in a dataset of previous observations. These unexpected behaviors are supposed to be suspicious and are usually called anomalies, or outliers. The goal of anomaly detection then is to highlight and flag suspicious events that differ significantly from the norm, as they could potentially indicate potential errors, fraud, attacks, and other events that require further investigation or intervention.

Anomaly detection is vital as anomalies in data translate often to important and critical incidents that can indicate potential risks, control failures or business opportunities. Anomaly detection has been extensively used in many fields, such as in healthcare, when detecting abnormal medical conditions from patient data or medical images, in finance and fraud detection, when identifying unusual patterns of credit card behavior that may indicate fraudulent activities and of course in cybersecurity, for example when detecting suspicious network traffic patterns that could indicate cyber-attacks.

As the amount of data used inside organizations is inconceivably high, it is impossible to manually monitor all systems for attacks. Anomaly detection allows organizations and individuals to automatically see imperceptible events or data points that show a significant deviation from normal operating patterns. When the system that employs anomaly detection finds outlier data, it alerts administrators of the event who may then take measures to handle that incident. Therefore, data points that are classified as anomalous are a critical aid to teams that try to find the source of security issues as fast as possible and defend their systems against potential threats.

At a high level, an anomaly is a pattern that does not conform to an expected normal behavior. Therefore, the goal of anomaly detection is to create a region or perception representing normal behavior and to classify any patterns that do not comply with that perception as anomalies. Although this seems like a straightforward approach, there are several factors that make this process challenging:

Firstly, the notion of an anomaly is domain-dependent and therefore differs across applications. To illustrate, a slight fluctuation of the heart rate of a patient might be classified as an anomaly while fluctuations in the stock market should be considered as normal. Thus, the techniques that were developed for a specific field do not guarantee satisfactory performance in other fields.

Secondly, it is difficult to define the region that completely and accurately encompasses every possible normal behavior. It is common that the notion of normal behavior changes over time and that the current perception may not be representative of future normal behaviors. Additionally, the boundary between normal and anomalous behavior is not clear and thus normal behaviors may be classified as anomalous and vice versa.

Third, the availability of labeled data for training and validation of anomaly detection models is a major issue. As will be seen later, some anomaly detection methods need data that are labeled, that is data that define which instances are normal and which are anomalous.

In conclusion, the anomaly detection problem is not easy to solve. Most of the existing anomaly detection methods are specialized in specific fields and solve specific instances of problems which are determined by the domain that the abnormalities need to be detected. Some of these methods are briefly described below, with focus being put on Auto-Encoders, the anomaly detection method used in this work.

2.3.1 Anomaly Detection as Learning Normal Behavior

Many anomaly detection methods revolve around the establishment of a baseline of normal and expected behavior. Therefore, the strategy for most approaches is to first model normal behavior, and then use this knowledge to identify deviations or abnormalities. This usual approach consists of a loop comprised of two main steps, the training step, and the test step, which introduces the concept of threshold-based detection.

The first step, the training, concerns itself with building a model of normal behavior using previous observations. Depending on the anomaly detection method chosen and the characteristics of the dataset, the training data may include both normal and abnormal data, or only normal data points. Using this data, the model will eventually hold a learned representation of normal behaviors and patterns, and it will be able to produce a metric that represents the measure of deviation from normal behavior.

The measure that is produced by the model is used to define how anomalies are reported. This value can generally be of two types: scores and labels. Scores represent the degree to which an instance is considered an anomaly and usually a threshold is chosen to select the scores that may be indicative of an anomaly. The threshold corresponds to a value (or a range of values), where scores above it (or outside of it) are identified as anomalies and below it (or inside it) as normal. Techniques that deal with labels assign a tag, normal or anomalous, to each instance without retaining any ordinal information between anomalies. Essentially, many scoring-based techniques transform the outputs to a label, since the introduction of thresholds creates two sets of instances, normal and abnormal.

During the second step of the anomaly detection loop, the model uses the concept of thresholds and using the score produced by the model it classifies each observation as anomalous or normal. The threshold concept is vital for the performance of the model as it provides the analysts with a way to manually tune the sensitivity of the system and its capacity to alert for anomalies. Usually, the higher the threshold the more correctly it will classify normal behaviors, but it may not capture anomalies successfully, especially when the differences between normal and abnormal behavior are not obvious. On the other hand, the lower the threshold, abnormal behaviors will be detected easier, but more normal instances may be regarded as anomalous, flooding the analysts with unnecessary alerts. Hence, the choice of the threshold is a vital step in developing anomaly detection systems and tuning their performance.

Although most anomaly detection methods follow this general approach and intuition, they differ with respect to their normal baselining methods and their outputs.

2.3.2 Taxonomy of Anomaly Detection Methods

Anomaly detection methods can be classified in various ways. There are many anomaly detection methods, like methods that rely on the statistical analysis of the data or on machine learning algorithms to automatically learn the patterns of normal behaviors. Because in this work I employ a machine learning-based anomaly detection method, in this section I review only the different types of techniques that use ML.

Overall, these approaches can be three different types of machine learning-based anomaly detection methods and they are categorized based on the existence of labels in the dataset used.

Supervised Anomaly Detection involves training a binary or multi-class classifier, using labels of both normal and anomalous data instances. Classifiers work best when the training dataset does not suffer from *class imbalance*, a phenomenon that occurs when the samples of one class dramatically outnumber the samples of the other. However, in anomaly detection tasks and specifically in cybersecurity applications there is lack of labeled datasets, since anomalous data are very rare.

Semi-supervised Anomaly Detection remediates the issue of class imbalance since it doesn't need labels from both classes to train. Such techniques, leverage the abundancy of the existing labels for only one class, usually what is considered normal, to separate outliers. As the labels of normal observations are far easier to obtain, semi-supervised techniques are more widely adopted that supervised ones. One common way of these algorithms, such as Auto-Encoders, is to train them on dataset that include no anomalies. Assuming enough training samples, semi-supervised algorithms learn correct representations of normal data points only, resulting in low loss values, and produce high loss values for anomalous inputs. Although many interpretations of semi-supervised techniques exist, this interpretation is the most relevant to this work.

Unsupervised Anomaly Detection techniques do not rely on labels in the training dataset. They usually assume that they contain instances of both classes, however anomalies occur rarely in the dataset. but it is not clear what each instance corresponds to. As it is not clear what instances are normal or outliers, unsupervised methods depend on the intrinsic properties of the data instances to detect anomalies. For example, by learning the distribution of some data, unsupervised detection identifies anomalies as those data points that lie in a given percentile or differ otherwise from that learned data. Nevertheless, in an unsupervised setting, both anomalous and normal data are assumed to be included in the train dataset but in different ratios.

Auto-Encoders, the model that is used throughout this project, are not clear to which category they belong and depending on the context they can share some characteristics of both categories. Auto-Encoders are not trained on labeled datasets, but the datasets are assumed to have only normal instances and their outputs are aimed to be the same as their inputs. For this reason, the categorization of Auto-Encoders is not clear. However, using the taxonomy presented, Auto-Encoders will be assumed to be a semi-supervised technique. As will be seen shortly, in this context an Auto-Encoder will be trained solely on normal data and their inability to correctly model abnormal inputs will be used to detect an anomaly.

2.3.3 EVALUATION METHODS

As was explained, when building anomaly detection systems, it is expected to deal with datasets that suffer from *class imbalance*. The most intuitive way to evaluate the performance of a model is to calculate its accuracy, which is the percentage of its correct predictions. But in such models, accuracy is not sufficient and may provide misleading results and lead to poor detection performance.

Although these systems may be accurate when classifying normal examples, they will perform poorly when classifying anomalous data. For instance, suppose a dataset that contains 100 input samples, of which 95 are normal and 5 are abnormal. A model that uses accuracy as its evaluation metric and classifies all samples as normal will result in a high overall accuracy of 95%, even though it didn't classify any of the anomalies correctly.

Of course, this behavior is unwanted in any classification task and especially in applications where the correct classification of abnormal instances is vital. In anomaly detection systems and particularly in cyber security, it is essential that firstly, the abnormal behaviors are captured and secondly, the normal behaviors are not mischaracterized as abnormal. Therefore, it is obvious that there is the need for the creation of other metrics that give a better picture of the skill of the detection model.

In the context of anomaly detection, Positives refer to instances classified as anomalies and Negatives refer to instances classified as normal. Although this contradicts intuition from other fields, it helps to remember that the goal of anomaly detection is to detect the *existence of anomalies*, and so a positive refers to the existence of an abnormality. Generally, normal instances correctly characterized as normal are referred to as True Negatives, normal instances mischaracterized as anomalies are referred to as False Positives, anomalous instances mischaracterized as normal are called False Negatives and finally anomalous instances correctly classified as anomalous are called True Negatives. All these are usually aggregated in the matrix in the table below, also called as a *Confusion Matrix*:

	Predicted Class	
True Class	True Positives (TP)	False Negatives (FN)
	False Positives (FP)	True Negatives (TN)

Table 1: Confusion Matrix

From these values, other metrics can be created which are more suitable for the task at hand. While there are many metrics that can be inferred from this table, for this work I focus my attention on two metrics that provide information solely on the incorrectly classified observations. This was deemed prudent, as the anomaly detection model developed should be evaluated based on its generation of False Positives, and its failure to detect malicious activity, hence the False Negatives.

The True Positive Rate (TPR), or Sensitivity, or Recall is the rate of the anomalies correctly identified as anomalies and the True Negative Rate (TNR), or Specificity is the rate of the normal samples predicted to be normal.

The False Negative Rate (FNR), or Miss Rate, refers to the percentage of the anomalies that are perceived to be normal. It is an important metric as it measures the failures of the system to capture anomalies. A high FNR leads to a poor detection model, allowing attacks to proceed without raising any alarms.

The False Positive Rate (FPR), or Fall-Out, refers to the proportion of normal behaviors mistakenly classified as anomalies. This metric is also significant as false positives generate alerts when there is no malicious activity. A high FPR will flood the security teams with false alarms and will hinder their ability to investigate incidents that indeed require their attention.

The formulas of these metrics are presented below:

$$FPR = \frac{FP}{N} = \frac{FP}{(FP + TN)} = 1 - TNR$$

Equation 1: False Positive Rate (FPR) formula

$$FNR = \frac{FN}{P} = \frac{FN}{(FN + TP)} = 1 - TPR$$

Equation 2: True Negative Rate (TNR) formula

Having introduced the nature, objectives, and methods of anomaly detection, in the next sections I present Auto-Encoders, which is the anomaly detection method that I use to build the detection system for this work.

2.3.4 AUTO-ENCODERS

Auto-Encoders are a type of neural network used in many applications including anomaly detection. Auto-Encoders fall within the family of encoder-decoder models and are designed to learn a low-dimensional and compact representation, called the "encoding" or “latent vector”, or “latent space representation” of some input data, and subsequently how to reconstruct the input from this representation. This latent vector aims to capture the essential features of the input data, any correlations, and other existing structures between the input features. Therefore, they are a special type of neural networks where the input is the same as the output, or more accurately, the output is very similar to the input.

Because neural networks are capable of learning nonlinear relationships, Auto-Encoders can be thought of as a more powerful generalization of Principal Component Analysis (PCA), which is a dimensionality reduction method designed to handle linear data. As most problems though are nonlinear, Auto-Encoders should be preferred for an accurate representation of nonlinear data.

An Auto-Encoder consists of three components: an encoder, a bottleneck, and a decoder. The encoder learns how to compress the input into a lower-dimensional latent space representation (bottleneck) and the decoder is tasked to reconstruct the original input using only that encoding.

Because the goal of the model is to successfully learn a compact representation of the input data and to learn how to effectively reconstruct the original input, the model is trained by minimizing the reconstruction error, which is the difference between the original input and the output produced by the decoder.

To use an Auto-Encoder for anomaly detection, the reconstructed version of the input is compared to the original input. If those two differ significantly, then the input is considered anomalous in some way. To comprehend better their application in anomaly detection tasks, it is important to understand that the model learns how to encode the training data and the patterns observed between them. So, when an encoder is exposed to data that differ significantly from the training data, that is data that do not exhibit the same trends or do not follow the same distributions to the training data, the model will not be able to successfully reconstruct the input. This unsuccessful reconstruction will result in a high reconstruction loss, which if it is above a specific threshold, can be inferred that that particular input corresponds to an anomaly.

2.3.5 Architecture

As mentioned before, Auto-Encoders consist of three parts: an Encoder, a Bottleneck, and a Decoder.

The first part, the *Encoder*, is tasked with the compression of the input data into an efficient encoding. It is essentially a fully connected Artificial Neural Network that typically consists of one or more hidden layers. It gradually reduces the dimensionality of the data by decreasing the number of neurons in each layer, extracting relevant features and any internal patterns along the way. By tuning the number of layers and the number of neurons in each layer we can modify the expressive capabilities of the model and its effectiveness to compress the input data to a lower-dimensional representation. The last hidden layer of the encoder connects to the Bottleneck, although many implementations consider the encoding to be last layer of the encoder.

The next component, the *Bottleneck*, consists of only one layer and has the smallest number of neurons, effectively representing a vector in a latent space. This bottleneck holds the encoding, or the latent space representation of the input data and it should include a low-dimensional and compact representation of all the input data's intricacies, interactions, and underlying structures. The number of neurons in the bottleneck is an important parameter of the model as it can control the amount of information that the encoding retains about a specific instance.

The third part, the Decoder takes the encoding from the encoder and attempts to reconstruct the original input from this compressed representation. It is also a neural network and is symmetric to the encoder, with hidden layers that gradually expand the dimensionality of the latent vector to match the original data's dimensions.

2.3.6 Training an Auto-Encoder

To use an Auto-Encoder for an anomaly detection task, the dataset does not need to be labeled and is assumed to contain only normal instances. To reiterate, Auto-Encoders learn how to compress the data based on attributes discovered from data during training. Thus, these models are typically only capable of reconstructing data similar to the class of observations which the model used during training.

Therefore, during training the Auto-Encoder learns how to model what a normal behavior looks like. The model takes as input normal instances and the encoder learns how to efficiently compress that information into an encoding. Subsequently, the Decoder takes that latent vector and learns how to successfully reconstruct the original input from that compressed representation. To guide the model to learn how to correctly reconstruct the original input, it is trained by minimizing the reconstruction error, which is the difference between that reconstruction and the original input. After going through the entire dataset and enough iterations, the Auto-Encoder will have learned how to efficiently create an encoding from the input data and how to reconstruct the input, leading to low reconstruction errors.

2.3.7 Anomaly Scoring

Because the model was trained solely on normal observations, it will learn how to compress and reconstruct data that follow the same distributions and patterns to normal behaviors. Therefore, as the Auto-Encoder attempts to reconstruct the input, it does so in a manner that is weighted toward normal samples.

This way, when a normal sample is given as input, the model will know how to reconstruct similar data, resulting in a low reconstruction error. On the other hand, if an anomalous instance is given to the model, it will fail to accurately regenerate the input, leading to a high reconstruction error, which means that the input differs significantly from the established baseline of normal behavior.

That reconstruction error, that is the difference between the output of the model and the original input can vary. Usual loss functions used for numerical outputs are the Mean Squared Error, and the Mean Absolute Error, but other functions can be used such as the Mean Squared Logarithmic Error, which as will be shown later, was used in this work.

Finally, after obtaining the reconstruction error between the output and the input of the model, we can use various thresholds to classify each instance as anomalous or normal.

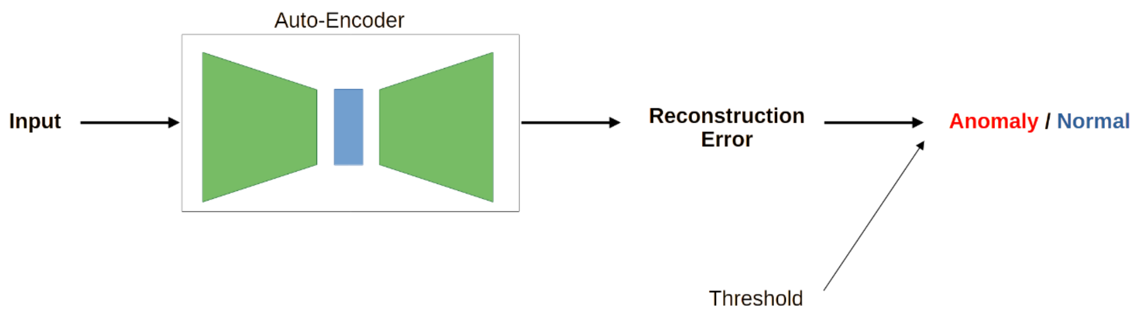


Figure 2-1: Classification of events as anomalous or normal using an Auto-Encoder

3 LITERATURE REVIEW

Static analysis can provide a great first defense measure for systems. Since it doesn't use any behavioral features but attributes that can be extracted by the files of the malware and their contents, such as file signatures, permissions, functions and more, it can stop ransomware before they are executed. doi={10.23919/ICACT.2018.8323680}} use static analysis to analyze the notorious WannaCry ransomware, to uncover its functions and processes. Additionally, DOI: 10.1109/RTEICT49044.2020.9315672 have used machine learning to classify ransomware-affected executable files from benign files. Although these works can enhance knowledge bases with many Indicators of Compromise (IoC) and be used for the prompt detection and prevention of attacks, static analysis is ineffective against novel and modern ransomware that can change their characteristics to evade such defenses. Static analysis is mostly effective against known malware and ineffective against polymorphic and metamorphic ransomware. Since modern ransomware can change their static characteristics by employing obfuscation methods https://unit42.paloaltonetworks.com/unit42-ransomware-locky-teslacrypt-other-malware-families-use-new-tool-to-evade-detection/, static analysis is insufficient to identify a program's type and objectives.

Other works have focused on the creation of systems around the analysis of the *API calls* made by programs. Ransomware have been observed to interact with Windows APIs, which are interfaces through which Windows offers a range of functions to applications. For example, File Scanning APIs are used to navigate the operating system's file system and Encryption APIs to perform cryptographic functions during the encryption of data https://n1ght-w0lf.github.io/malware%20analysis/ryuk-ransomware/. For that reason, API usage has been used to detect ransomware. https://www.sciencedirect.com/science/article/pii/S1319157820304122 have proposed PEDA, a Pre-Encryption Detection Algorithm that can detect crypto-ransomware at the pre-encryption stage, when no encryption has been done. It intercepts all the API calls made by processes until the first file is encrypted by the ransomware and trains a learning algorithm to discern goodware API usage from ransomware. Additionally, https://doi.org/10.1016/j.jnca.2020.102753 have managed to also use machine learning using Windows API traces to discriminate ransomware behavior, resulting in early detection. Although these works yield strong results, focus solely on the API calls made is ineffective against many ransomware, as attackers will be prompted to write and statically link DLLs, avoiding interaction with the provided Windows APIs and thus bypassing detection. Therefore, while API usage can be indicative of some ransomware strains it should be used as a part of a larger feature set.

Storage interaction has been seen to be able to detect ransomware infections. doi={10.1109/IOTSMS48152.2019.8939214}} have constructed a dataset of five features using ransomware and benign I/O logs. They also trained a K-Nearest-Neighborhood (KNN), a Support Vector Machine (SVM) and a Random Forest classifier to achieve an F-measure rate of 98%. While these features can generally detect ransomware, they are not able to detect them in their early stages. https://doi.org/10.1145/3243734.3278491 designed a ransomware-aware buffer management policy and an access-pattern-based detector that allows encryption-based ransomware behavior identification and even ransomware that exploit kernel vulnerabilities. Although detection at the encryption stage is important as well, early detection is crucial for a timely defense. Since also, early detection is the primary focus of this work, such features should be used in cooperation with other behavioral features.

Network activity is an important aspect of ransomware attacks, during communication with the C&C servers, data exfiltration, discovery of target files, and self-propagation. https://doi.org/10.1016/j.jnca.2018.09.013 have developed REDFISH, a system that detects ransomware strains that encrypt files in network shared volumes. By analyzing traffic that was passively monitored by a network probe and related to basic behavior of reading, writing, and removing files, they managed to detect 99% of the cases of ransomware, before 10 files are deleted, with a small false positive rate. Although this work presents strong results and is able to detect attacks with a very small impact on the victim's files, it is ineffective against ransomware that do not target network shared volumes. In addition, https://doi.org/10.1145/3180465.3180467} utilized Programmable Forward Engines (PFEs) to collect per-packet data by monitoring networks at high-rates. Using that data, they trained a Random Forest classifier to identify infected machines contacting C&C servers before encryption takes place. It is noteworthy that they did not use deep packet inspection to extract their features but relied solely on the metadata and thus unencrypted information in the packet headers to construct their feature set. This is an important advancement, as previous works, such as http://arxiv.org/abs/1611.08294, relied on packet inspection and were inefficacious against ransomware that utilize encrypted traffic such as HTTPS to communicate with the C&C servers. Thus, this research can influence the creation detection systems by using features that can be extracted from the metadata and statistics of the traffic flows.

Moreover, *deception-based approaches* have shown that they can help strengthen the early detection capabilities of ransomware attacks. https://doi.org/10.1016/j.cose.2017.11.019 created R-Locker, a system that detects ransomware attacks by strategically placing decoy files around the target environment. Additionally, it thwarts ransomware by blocking it once it starts reading a "honeyfile". Although tripwire files and folders can identify an ongoing attack, they do not provide a guarantee that the malware would attempt to invade these areas, and therefore bypassing this defense. Therefore, since the non-existence of alerts does not indicate an active attack, honeypot-based approaches should be a supplementary measure employed along with other more wide-ranging systems. Although this work does not employ a honeypot-based approach, it is important to analyze such efforts for completeness and to emphasize their limitations and their unsuitability for this study.

One other notable mention is the use of side channel data to detect ransomware attacks. https://s2.smu.edu/~mitch/ftp_dir/pubs/ftc17.pdf proposed a technique to detect encryption and thus ransomware attacks based not on the characteristics of the data affected, but on hardware physical sensor data streams, such as CPU temperature readings, per core CPU load, power consumption, and more similar features motivated by the labor and power intensive operations that ransomware are based on to perform their malicious activities. Using that data, they trained a logistic regression classifier and detected ransomware attacks very early in the encryption stage, resulting in low data loss. However, this approach has two major limitations. Firstly, encryption may be an integral part of many legitimate activities, and the detection of encryption does not necessarily imply ransomware attacks. Secondly, reliance on physical sensors renders the detection system highly machine dependent due to their different capacities and environments. Consequently, such approaches are not generalizable and may produce large quantities of false positives and should therefore be avoided.

https://arxiv.org/abs/1609.03020 understood that the strength of detection systems lies in the diversity of the features used. They explored the strengths of dynamic analysis and introduced EldeRan, which monitors a set of actions performed by applications in the first phases of installation, such as Registry Keys operations, API usage statistics and File System and Directory operations and strings embedded in the binary file of the executables. Using features belonging to those categories, they trained a regularized logistic regression classifier, that correctly classifies the vast majority of the cases encountered and produce a small number of false positives alerts, showcasing the strengths of dynamic analysis. While 1.6% false positive rate is relatively low, if we consider that an enormous number of events happen throughout the day, 1.6% generates a huge number of alerts that are eventually not malicious, flooding the security analysts with alerts and reducing their efficiency. However, this work presents promising results, and it shows that the combination of diverse features belonging to many stages of ransomware attacks can out-perform traditional detection mechanisms, provide meaningful improvements to the defense infrastructure of organizations, and serve as the foundation for the exploration of more features and techniques that can detect ransomware and reduce the number of false positives generated.

Auto-Encoders have widely been used in anomaly detection tasks in the cyber security domain. doi={10.1109/ACCESS.2022.3155695}} have used Auto-Encoders to detect malware by converting gray-scale images of software samples. Feeding these images to the Auto-Encoder they achieved an accuracy and an F-score of 96%. In addition, https://api.semanticscholar.org/CorpusID:44055419 employed Auto-Encoders to create a Network-based Intrusion Detection System, which resulted in a 99% True Positive Rate. Adding to those works, http://dx.doi.org/10.1155/2019/8195395 detected malware in IoT environments by extracting API calls and building behavioral graphs. Using Auto-Encoders they succeeded in learning the higher-level semantics of malicious behaviors from the behavior graphs and further increase the average detection precision by 1.5% in comparison to other related works. Finally, doi={10.1109/WTS.2018.8363930}} introduced an Auto-Encoder-based network anomaly detection method, to detect network anomalies in the NSL-KDD dataset, a modification of a dataset commonly used to train and evaluate intrusion detection systems. From these works, it is apparent that Auto-Encoders are a powerful anomaly detection method that can be used to identify malware. This way, this work strives to expand the use of Auto-Encoders in the anomaly detection domain and specifically in ransomware detection tasks, as they show promise that they can successfully be used for the detection of such threats.

In summary, existing research have focused mostly on the general classification of ransomware behaviors without taking timing into account or have focused on a confined set of features to rapidly detect ransomware that are either easily evaded or produce many false positives. Other works have created systems that are not easily generalizable and have no prospects in diverse environments. However, there are some systems that are promising and can be used as a foundation and inspiration for the creation of more robust and sensitive systems that can detect ransomware attacks before they start encrypting or exfiltrating the victim's data. It is noteworthy that during the analysis of these works, it was observed that machine learning methods are ubiquitous. This fact shows and validates the promise and the capabilities that ML has in cybersecurity. This work aims to produce such a system, by combining and utilizing the knowledge and experience of previous works, the capabilities of Auto-Encoders, and to create a diverse feature set that captures ransomware behaviors in many stages of its lifespan. Over the next chapters, I will describe the methodology I followed, what features will be used to devise such a system, what techniques will be used and finally how this system can detect ransomware attacks before they have harmed the victim's assets, while minimizing the number of false positives generated.

4 METHODOLOGY

It is clear that the defense industry and research community need a system that can detect ransomware attacks as early as possible, while taking special care of false positive alerts. The main objective of this work is to construct a system that can detect ransomware attacks on a device in the earliest stages of the ransomware kill chain, and simultaneously generates a minimal amount of false positive alerts.

To achieve that, I propose firstly to create a comprehensive dataset comprised of features that correspond to the various stages of a ransomware infection. Using that dataset, I can model the normal operation of a system and use that to detect significant deviations and classify them as ransomware attack incidents. To make that classification, I further propose to train a machine learning-based anomaly detection model on those data, specifically an Auto-Encoder, as it has shown its strong capabilities in cybersecurity and anomaly detection tasks.

To create that dataset, I suggest monitoring a target device for a significant period, so I can capture the normal behavioral patterns of processes. As the features that will be extracted to implement the detection system are not available in any other dataset due to their needed granularity, deep view of the system's operation and the flexibility required for feature experimentation and exploration, it was not possible to use an existing dataset.

Also, I propose identifying the various points in the abstract ransomware kill chain and their effects on a machine that are indicators of a ransomware attack. Combining them I will create an accurate depiction of the normal operation of the machine and deviations from it will thus indicate a potential ransomware attack.

Subsequently, an Auto-Encoder should be trained on those data to establish a baseline for the normal behaviors of processes and mathematically uncover their patterns and trends.

To evaluate the detection capabilities of this model, I advise to develop a series of experiments and deploy a custom ransomware that performs the most important steps taken by ransomware as presented in the ransomware kill chain. By deploying the trained detection model and the ransomware in the test device, I can monitor the behavior of the system's processes and use the inferred features to test if the model can detect the ransomware infection, at what stage of the attack it is captured and finally the number of false positives and false negatives that are generated.

The primary focus of the experiments should be the evaluation of the detection model as a whole, the capabilities of the model to detect attacks but also to not misclassify normal behavior as anomalies. Supplementarily, the experimentation phase should also test the performance of the model against ransomware that employ some defense evasion techniques. Although stealthy ransomware is not the primary objective of this work, it is beneficial to include experiments that stretch the capabilities of the model to provide a comparison of its performance against more sophisticated ransomware.

4.1 DATASET

In this section I outline the structure of the dataset, the features that should be and are used and describe the ways that they were collected to create the final dataset.

The general goal of the system is to detect ransomware based on the activities performed by the processes of the system. Before I continue, it is important to state that I assume that the ransomware attack is performed by a single process, meaning that all the steps of the attack are performed by the same process. Although, this might not be the case in some attacks detected in the wild, this work focuses on ransomware that utilize a single process. Nevertheless, I didn't entirely overlook the cases where an attack cooperates with other processes, and I include some features that take these scenarios into account.

4.1.1 DATASET STRUCTURE

Concerning the structure of the dataset, I propose that it should have a tabular structure, where each entry corresponds to the observed activity of a single process for a specific period. This period can be of different ranges, varying from fractions of seconds to even minutes or hours. However, as ransomware may perform their tasks fast, large ranges will fail to provide the granular view required to break down the steps of ransomware attacks and may result in an unsuccessful early detection. On the other hand, if the time range is small, the processes may not have enough time to exhibit significant volumes of activity, such as encrypting many files or exfiltrating large quantities of data, rendering ransomware and normal activity almost indistinguishable.

It is necessary to have a range large enough to capture noticeable process activity volumes, but also small enough so it can detect the distinct steps of an attack as it progresses through its kill chain. Therefore, I chose to aggregate the observed activities of each active process every *second*, enabling the detection system to have a detailed view of the activities of each process and to capture the attack in its various early stages.

Therefore, each entry of the dataset will hold the observed activity of *each active process for every second* that it is active. The following table gives an outline of the structure of the dataset:

		Feature_A	…	Feature_Z
sec 0	Process_A			
sec 0	Process_B			
sec 0	Process_C			
sec 1	Process_A			
sec 1	Process_B			
sec 1	Process_C			

Table 2: Dataset Structure

Finally, all the datasets that are used for the training, validation and testing of the detection system will have this structure. The system should be monitored for a number of days, and for each second of the monitored period the activities of all the processes will be inspected to populate the various datasets.

4.1.2 Dataset Creation and Data Collection

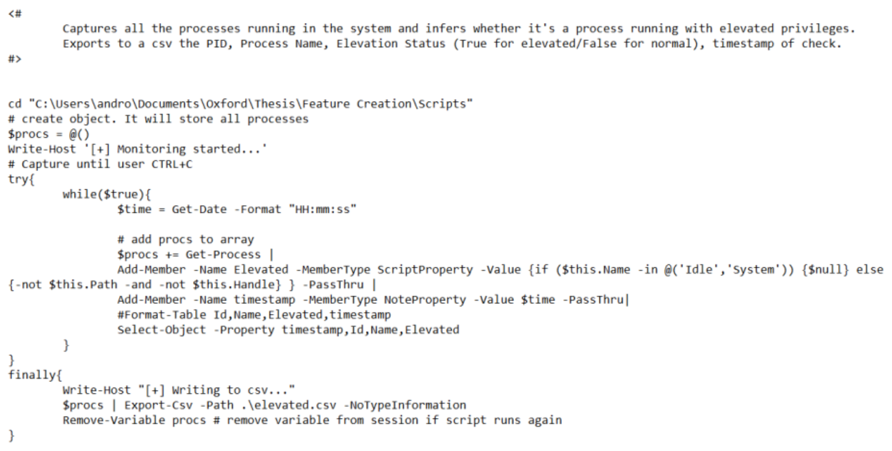
To create the dataset, I must first think of the ways that the behavior of ransomware may deviate from that of normal processes. Despite their limitations, the previous works presented have been instrumental, as they have helped uncover the potential points in a ransomware infection that can be indicators of compromise and provide intuition to explore more features.

Firstly, as the features used during static analysis of ransomware have significant drawbacks, I pursue a purely behavior-driven detection method. In this work I focus on behaviors and patterns that are generated by ransomware during execution from the first stages of an attack until the attack is completed. Secondly, I focus on the steps taken by ransomware once they are inside a system and are executed, so phases such as Reconnaissance, Weaponization and Delivery are overlooked. Moreover, during the exploration of those features I did not focus on ransomware that self-propagate as they are not a crucial characteristic of ransomware. Similarly, C&C communication is not an indispensable part of a ransomware attack, so I chose not to focus on features that detect such traffic except from data exfiltration activities.

To gather the necessary data, the test system should be monitored for a specific period across different time contexts. In this work I used the Process Monitor (procmon) utility, an advanced tool for Windows that monitors and records changes made in the system by all running processes in terms of file system, registry, network, and general process activities. Additionally, I utilized a self-developed script to supplement Process Monitor and infer the privileges of active processes.

Initially, because a good anomaly detection system should detect anomalies based on the time context as well, I keep track of the time of day that a process is running. If the time is between 09:00 to 17:00 I record this as a working hour and any time outside of that I record it to be a non-working hour.

Many actions performed by ransomware require elevated privileges and for that reason, during the first steps of the execution ransomware processes try to escalate their privileges. While it is hard to enumerate and capture all methods that a process might gain such permission, I propose to monitor the status and abstractly infer the privileges of each active process on the system. That way, regardless of what method a process might leverage to escalate its privileges, the change of process's privileges will be noticeable by the change of the process's status. To keep track of the privileges of each process I created the PowerShell script shown in the figure below, that iteratively (every second) enumerates all the processes running in a system and their privileges, which are stored in the binary feature "ELEVATED", where 0 means that the process does not run with elevated privileges and 1 implies the opposite.



```
<#
        Captures all the processes running in the system and infers whether it's a process running with elevated privileges.
        Exports to a csv the PID, Process Name, Elevation Status (True for elevated/False for normal), timestamp of check.
#>


cd "C:\Users\andro\Documents\Oxford\Thesis\Feature Creation\Scripts"
# create object. It will store all processes
$procs = @()
Write-Host '[+] Monitoring started...'
# Capture until user CTRL+C
try{
        while($true){
                $time = Get-Date -Format "HH:mm:ss"

                # add procs to array
                $procs += Get-Process |
                Add-Member -Name Elevated -MemberType ScriptProperty -Value {if ($this.Name -in @('Idle','System')) {$null} else
{-not $this.Path -and -not $this.Handle} } -PassThru |
                Add-Member -Name timestamp -MemberType NoteProperty -Value $time -PassThru|
                #Format-Table Id,Name,Elevated,timestamp
                Select-Object -Property timestamp,Id,Name,Elevated
        }
}
finally{
        Write-Host "[+] Writing to csv..."
        $procs | Export-Csv -Path .\elevated.csv -NoTypeInformation
        Remove-Variable procs # remove variable from session if script runs again
}
```

Figure 4-1: PowerShell script used to get the privileges of all running processes every second.

Progressing in the ransomware kill chain, ransomware try to gain persistence by modifying the Windows Registry AutoRun keys. Specifically, ransomware may attempt to modify the following subkeys:

- “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run”
- “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce”
- “HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run”
- “HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce”

From these keys, the first two require elevated privileges to be modified, while the last two can be modified with normal user. To capture that activity, I monitored *any* interactions with these AutoRun keys and recorded them in the feature “AUTORUN_EDITS”.

Some ransomware may rely on other processes to add the main malware to the AutoRun keys or may want to trigger the execution of the ransomware every time a user logs on the device. For this reason, it is prudent to monitor any interaction with the keys, not just write or set operations. In such cases, the process that modifies the registry and the main body of the malware may have different Process IDs and their cooperation cannot be easily traced. To accommodate that, I advocate to include another feature, called “EXISTS_IN_AUTORUN”, which states whether a process exists already in the Registry AutoRun keys. The “Command Line” column of procmon lists the command that was issued to execute the process. Therefore, by comparing the values of the existing registry entries and the command line listed in procmon, after stripping any command line arguments, I inferred which processes already exist as AutoRun keys.

Subsequently, ransomware try to terminate certain processes that inhibit file access, encryption or that generally relate to defense systems such as antivirus software, firewalls etc. To terminate other processes, programs use numerous tools, such as “TASKKILL”. Normal applications use such tools very rarely and such events should be included as a part of the feature set, that can differentiate normal from ransomware activity, using the binary feature “KILL_DEFENSE”. Processes that use that tool usually spawn another process, that will issue the “taskkill” command. However, this behavior can be captured by procmon, by inspecting any “Process Create” operations and then checking if the Detail column contains the “taskkill” string. Undoubtedly, this feature does not intend to capture all possible methods that ransomware use to disable the victim’s defenses. Instead, it aims to show that if such behaviors are observable, they can help detect attacks. For that reason, it suffices to show that for at least one such method, the consideration of “taskkill” is adequate.

Moving forward, ransomware are expected to traverse an abnormal amount of directories in a very short period of time when trying to discover files that they wish to encrypt. Therefore, the volume of directory traversal and file enumeration efforts should be monitored. When a process traverses directories and enumerated their files, it generates "QueryDirectory" operations. Therefore, I count the "QueryDirectory" operations in procmon's "Operation" column, which is then stores in the "QUERY_DIRECTORY" feature.

After locating the target files, ransomware usually will try to exfiltrate them to a remote location. This activity provides another point that can be used to detect an infection. By monitoring the volume of egress traffic initiated by a process we can detect data exfiltration efforts. As ransomware may try to mix their traffic with normal traffic to evade defense mechanisms I use a high-level feature, which does not depend on the protocols used by the attackers, (custom protocols, encrypted traffic, TCP/UDP etc.). Simply the volume of egress traffic can be used as a feature called "NET", which is extracted by filtering in procmon all the network operations that contain the word "Send" and summing the number listed in the Length attribute of the "Detail" column.

Moving forward in the kill chain, the use of encryption may also mean the usage of cryptography related DLLs. Such DLLs may be the "bcrypt.dll", the "crypt32.dll" and other DLLs that are part of the various Windows Cryptography APIs. As was explained earlier, attackers may bypass the use of such APIs by statically linking encryption DLLs. However, API and DLL monitoring should not be discarded and should be used only as a part of a larger feature set. To incorporate the detection of such activity, I keep track of any "Load Image" operations recorded in procmon and I check if any of the images loaded contain the string "crypt", as this string is frequent in all the cryptography-related libraries. If the use of such DLLs is observed this is recorded in the "CRYPT_DLLS_LOADED" feature.

When ransomware move on to the encryption phase, they can be detected using a variety of features. Although encryption itself is not easily traceable, one can infer the use of encryption by inspecting the side effects of certain activities. For instance, ransomware are anticipated to write to many files in a very short amount of time, especially since attacks usually speed up when they reach the encryption phase. Thus, one should track the number of files that a process writes to. However, not all file types should be considered in the calculation of that feature. Instead, a detection system should monitor only the file types that are common targets of ransomware attacks. Overall, ransomware aim to encrypt text documents, Microsoft Office and LibreOffice documents, images, videos, audio files, and compressed files, to compromise documents that may contain personal, private, and sensitive information of the victim or other indirectly linked stakeholders, and any other data and information that may disrupt the normal operation of the victim's activities. The comprehensive list of the extensions monitored by the system can be seen in the Table 3. These files that contain such information have more value to the attackers and can be used as a leverage against the victim to pay the ransom. To track the number of "interesting" files written by a process in a certain amount of time, I use the “Path” column’s entries to infer the file that was modified, and I count the number of “WriteFile” operations for each identified file.

TYPE OF FILE MONITORED	FILE EXTENSIONS MONITORED
Text Documents	txt
Images	jpeg, jpg, png, gif
Videos	mp4, avi, m4a, mov, mkv
Audio Files	mp3, wav
Microsoft Office/Libre Office documents	docx, pptx, xlsx, odt, ods, odg
Compressed Archives	zip, rar, 7z

Table 3: File types and file extensions targeted by ransomware and monitored by the detection system

Encryption tends to generate high-entropy data. In information theory, “Shannon Entropy” is a measure of the randomness of a variable and can indicate how “random” or “unpredictable” the data of a file are. This measure takes values between 0 and 8, indicating no or complete randomness respectively in the given file. Therefore, as encryption generates seemingly random sequences of data, it also produces on average high-entropy data. Thus, the average written entropy by each process can be used as a feature to infer encryption, additionally to the number of files that were modified by that process, by using the Shannon Entropy formula shown in the equation below for all the files that were identified in the previous feature. Finally, by averaging the entropy of all the files written by each process the “AVG_WRITTEN_ENTROPY” feature is populated. Here it is worthy to state that if a process did not write to any files during that period, its entry is filled with the default value of 4. This value was chosen as a value of 0 would heavily influence the model to perceive the “normal” entropy as close to 0, indicating absolutely no randomness. This behavior is unwanted as the usual entropy of normally used files should have some amount of randomness. Additionally, 4 is a suitable value as it also lies in the middle of the entropy’s possible range, which is very close to the mean of the dataset’s entropy feature as will be presented in a later section.

$$Shannon\ Entropy\ = -\frac{1}{N}\sum_{i=1}^{N} p(x_i)\, log_2\big(p(x_i)\big)$$

$$where\ N\ is\ the\ length\ of\ the\ sequence\ and\ p(x_i)\ is\ the\ probability\ of\ obtaining\ the\ value\ x_i$$

Equation 3: Shannon Entropy

Furthermore, I advocate that a ransomware may exhibit different writing patterns when writing to files. Specifically, the average number of write operations per file can be a sign of infection. One may understand that, if they think that a process may write many times to a file as they modify that file. On the other hand, ransomware are expected to write to files only once, when they write the encrypted data of the files, and never write any data to them again. Supplementarily to the previous two features, this can help distinguish normal from ransomware activity. Thus, this information is recorded in the feature “AVG_WRITES_FILES”, by calculating the average write operations made in the files identified in the feature above.

Finally, as it is typical for ransomware to hinder the recovery of the system and deleting backups, I propose a final feature “DELETE_BACKUP”, which states whether a process has tried to inhibit the recovery of the system after an attack. Similarly to the “KILL_DEFENSE” feature, ransomware may achieve that in a plethora of ways. However, for the same reasons, it suffices to include only one method in this feature. Thus, this feature aims to capture attempts by processes to disable the Windows Recovery Environment (WinRE), which allows the recovery of a system in many cases of grave malfunctions or attacks. This attribute is populated in an identical way to the “KILL_DEFENSE” feature, but instead of the “taskkill” command the “reagentc /disable” command is monitored using procmon.

Some observed activities should not disappear by the next observation period of the process. For example, if a process has interacted with some AutoRun keys at a specific time, the subsequent observations should not “forget” this action. Instead, I propose for the features “AUTORUN_EDITS”, “KILL_DEFENSE”, “DELETE_BACKUP” and “CRYPT_DLLS_LOADED” to have a memory of previous activities. This memory is given to a process, by adding to all subsequent samples, after the observation of a positive event, the positive event. For example, if a process edited the registry and time t=2, all subsequent samples t>2 of that process will have a value of 1, to indicate that previously that process edited the registry, regardless of the fact that during those periods no registry interaction was detected.

One reasonable question that one may raise is that all these features correspond to behaviors seen by benign processes as well. For instance, a web browser sends data to other hosts as a part of their normal operation and compression applications may write to files with a high data entropy, due to the nature of compression. While that is true in some sense, I advocate that while many applications may exhibit ransomware-like behaviors for some features, it is not expected that they will exhibit malicious behaviors for these features simultaneously. Therefore, the *combination* of all the features is what will eventually differentiate normal from suspicious behavior.

PUT TABLE THAT MAPS THE KILL CHAIN STEPS TO THE FEATURES USED

In summation, the final dataset is comprised of the following features:

FEATURE	DESCRIPTION	TYPE
WORKING_HOUR	The process runs during working hours	Binary
NET	Bytes sent by a process	Numerical
AUTORUN_EDITS	Interaction with the Autorun Registry keys	Binary
EXISTS_IN_AUTORUN	The process exists as an AutoRun program in the registry	Binary
CRYPT_DLLS_LOADED	The process uses any DLLs associated with cryptographic primitives	Binary
FILES_WRITTEN	Number of files written	Numerical
AVG_WRITTEN_ENTROPY	The average entropy of files written	Numerical
AVG_WRITES_FILES	The average number of write operations per file	Numerical
ELEVATED	Process running with elevated privileges	Binary
QUERY_DIRECTORY	The number of "QueryDirectory" operations performed by a process when enumerating the files in a directory	Numerical
KILL_DEFENSE	The process attempts to terminate processes related to defense mechanisms	Binary
DELETE_BACKUP	The process attempts to inhibit recovery of the system or delete backups.	Binary

Table 4: Features used in the dataset.

4.2 DETECTION MODEL

For a reliant detection system, equally important to the features used is the model that is deployed. In this work I propose to train an Auto-Encoder to learn a representation of the operation of normal applications. After the normal behavior has been baselined, that model will be used to infer if the behavior of a process is an anomaly.

Primarily, it is important to argue why a deep learning approach is preferred. Deep learning methods can handle multivariate and high-dimensional data reducing the required effort to individually model anomalies for each variable and aggregate the results. Additionally, these methods offer the opportunity to quickly model complex and nonlinear relationships in data, leading to high performance in anomaly detection tasks. Moreover, their performance can potentially scale with the abundance of training data, making them suitable for data-rich problems, such as this task. Although many anomaly detection models exist, I suggest that an Auto-Encoder suffices for the scope of this work and can produce the intended results.

As was explained earlier, an Auto-Encoder consists of two main parts, an Encoder, and a Decoder. The Encoder learns to encode the input data to an efficient representation of fewer dimensions and the decoder learns to reconstruct the input data from that compressed representation.

In this work, the Auto-Encoder will learn the patterns and trends of normally-behaving processes. By minimizing the reconstruction error during training, the Auto-Encoder will effectively encapsulate and baseline the normal behavior of the system, and it will be able to reconstruct accurately any other normal behaviors observed.

This way, when there is an ongoing ransomware attack the Auto-Encoder will not be able to reconstruct the original input from the latent representation, as it has been trained solely on legitimate instances. Generally, the output of the Auto-Encoder will be the reconstruction error of the input given. It is expected that normal inputs, that is samples that correspond to normal behavior, will have low reconstruction errors and anomalous inputs, will result in higher reconstruction errors. Using that reconstruction error, we can infer if the observed behavior corresponds to a normal operation of the system or an ongoing ransomware attack.

Since the output of the system is simply the reconstruction error of the input against the output of the Autoencoder, it is important to provide a threshold above which any observed sample will be classified as an anomaly. Although this is a process that will be described in a later section, it must be said that there are many choices for thresholds that can influence the efficiency of the model.

4.3 Establishing a Normal Baseline

Additionally to the features used and the chosen model, the quality of the training data is of vital importance. For the system to be reliable, the data collected must reflect a genuine use of the system where plentiful and realistic interaction of the user with the system occurs.

The target device should be monitored for a significant period while a user uses the device for normal daily activities. As the goal is to baseline the normal behavior, the user should perform a diverse set of activities that they usually do throughout the day, such as browsing the web, editing documents, viewing media, modifying files and any other legitimate operations.

Additionally, the device should be monitored across different times of the day, to capture the behavioral patterns in both working and non-working hours times. Generally, it is expected that during working hours, devices tend to experience an increased workload of diverse activities, as users are expected to interact heavily with their devices. In contrast, during non-working hours, users are anticipated to use their devices significantly less, generating less and more confined sets of activities. This will provide temporal context to the model and increase its expressive capabilities.

Lastly, it is imperative that while the system is monitored to baseline normal, the device is not infected with any kind of malware. As the Auto-Encoder will be trained to model only the normal behavior, the training data should not contain any samples extracted from malicious processes. Thus, it is crucial to ensure that the device that is used to create the dataset is not infected with malware, as otherwise the model will perceive malicious behavior to be part of the normal activities of the system.

4.4 Testing

The utility and performance of the created system can be shown only through a set of experiments. To show how effective the system is I propose to conduct a set of experiments on some test data and record its detection capabilities. Because the evaluation of the system should be done regarding both the number of false negatives and false positives generated, the model should try to correctly classify both normal and ransomware activity.

To simulate a legitimate use scenario, I first propose to deploy the detection system while the test device is used normally. This way, the system will be evaluated on whether it can correctly classify normal behaviors, and thus how many false positives it produces. To evaluate the core task of the system, a ransomware attack should be conducted on the machine and the trained model should try to classify each observed activity as normal or anomalous. This way, the system will be evaluated on both the timing of the detection, that is at which of the pre-encryption steps the attack was detected and by extension the false negatives generated.

Additionally, it is vital to test the detection system in different time contexts. Overall, I consider two different time contexts, one during which a lot of diverse and intense activity is expected, simulating the workload of a working hour in an organization or a personal computer, and another one that expects a significant reduction in the observed workload, corresponding to the non-working hours of an organization.

Hence, it is proposed that two sets of experiments are performed. In the first one, the system should attempt to detect the anomalous activities of the ransomware during working hours and in the second one the model should attempt to detect the ransomware if it is deployed during non-working hours. In both experiments, the device should be used normally to simulate the expected activities of these two different setups. Finally, the model should be evaluated firstly on how early it detected the attack, i.e. at which stage of the kill chain the attack was detected, or which actions went unnoticed and finally, on the number of false positive alerts and false negatives were generated.

It follows then that a test ransomware attack should also be developed and performed on the test system. This ransomware is described in the next section.

4.5 THE TEST RANSOMWARE

To test my model and its detection capabilities, it was essential to simulate an attack. For that reason, I chose to develop a ransomware which exhibits the major attributes of a ransomware attack. I developed this ransomware using Python 3.9, as Python offers a wide variety of tools useful for the development of a ransomware and interaction with the system. Of course, the ransomware could have been written in any other programming language, but Python offers great capabilities for the creation of such a script.

Since for this work, we are not interested in the first phases of a ransomware attack (Reconnaissance, Weaponization, Delivery) I focused on the steps taken by common ransomware from the Exploitation phase onwards, assuming that the malware exists on the victim's device and is being executed. Below, I describe the core actions taken by this ransomware to understand the way it works and other assumptions that I have taken.

First and foremost, the attack is conducted by a *single process*, without the help of any other child processes. It doesn't spawn any other processes and doesn't rely on the coordination between other processes that may have been delegated certain aspects of the attack.

Once the script is executed, it first tries to escalate its privileges by checking if it runs with elevated privileges. If it does not, then it tries to elevate them with a UAC prompt by directly asking the user. If the user grants them, the process is restarted with elevated privileges. A UAC prompt was chosen as the feature chosen to capture the privileges of the process is not specific to exploitation techniques or UAC prompt requests. On the contrary, it is technique-ignorant and generally checks if a process is running in elevated mode. This way, it can capture any technique used by an attacker to escalate the privileges of a process.

Having acquired elevated privileges, the script tries to gain *persistence* by modifying the Windows Registry and adding itself as an Autorun program. Specifically, it creates itself in the

"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Rans " key.

Later, it uses the "taskkill" command to terminate any processes that may be used for detection of the attack, such as Antivirus software, or inhibit the encryption of files.

Afterwards, it traverses all the directories of the system to locate target files. At this point it is worthy to note, that instead of traversing all the directories starting from the C:/ directory, the ransomware was configured to start the search from a certain directory that mimics the root of the directories. Starting from that directory it searches all the subdirectories in a depth-first manner, to locate all the files that it wishes to encrypt and the directories in which they are stored.

As many ransomware do, this ransomware does not blindly target files indiscriminately. On the contrary, it is configured to encrypt only a specific set of file types, by examining their extensions. Specifically, the ransomware targets the files that are listen in the Table 3 .Overall, the ransomware aims to encrypt documents, multimedia files, and compressed archives, to compromise documents that may contain personal, private and sensitive information of various stakeholders, and any other files that may disrupt the normal operation of the victim's activities. Additionally, since (as will be explained shortly) the ransomware plans to exfiltrate the data to a C&C and threaten the victim to publish them unless the ransom is paid, files that contain such information have more value to the attackers and can be used as a leverage against the victim.

The malware then proceeds to gather information about the device, such as its OS, its version and release, its hostname, and its architecture, to send to the C&C, via a simple custom protocol that just sends the data as plain text. Although this information will not be used in any other way by the C&C to tailor the attack, it is gathered to mimic the communication of the victim with the C&C.

Shortly after the victim transmits the victim information to the C&C, it generates a symmetric key using the AES 128-bit cipher in CBC mode utilizing the Fernet python package. This key is the key that will be used to encrypt all the target files. Moreover, AES was used as the preferred cipher as it is a strong and symmetric cipher, allowing fast encryption of a large quantity of files, directly corresponding to modern and genuine ransomware observed. For the purposes of this ransomware, it suffices to use just symmetric encryption, as hybrid encryption is not necessary to create mechanisms that inhibit the discovery of the key during recovery after the attack.

When the key is generated, it is sent to the C&C server so it can provide the decryption key once the ransom is paid, and the victim can decrypt their assets. When the key is transmitted, the ransomware connects with another C&C server that administers data exfiltration and sends all the target files to it. This server is a simple HTTP(S) server, instead of communicating over a custom protocol (as the previous one). Data exfiltration to an HTTP/S server is a very common method chosen for data exfiltration attacks because it can give a significant amount of cover as devices are expected to communicate frequently using such protocols. Specifically, data exfiltration over HTTP/S can blend with normal browsing behavior of the users, or Web API usage from legitimate applications. Therefore, the ransomware connects to the HTTP server on the alternative HTTP port 8080 and send the files with HTTP POST requests.

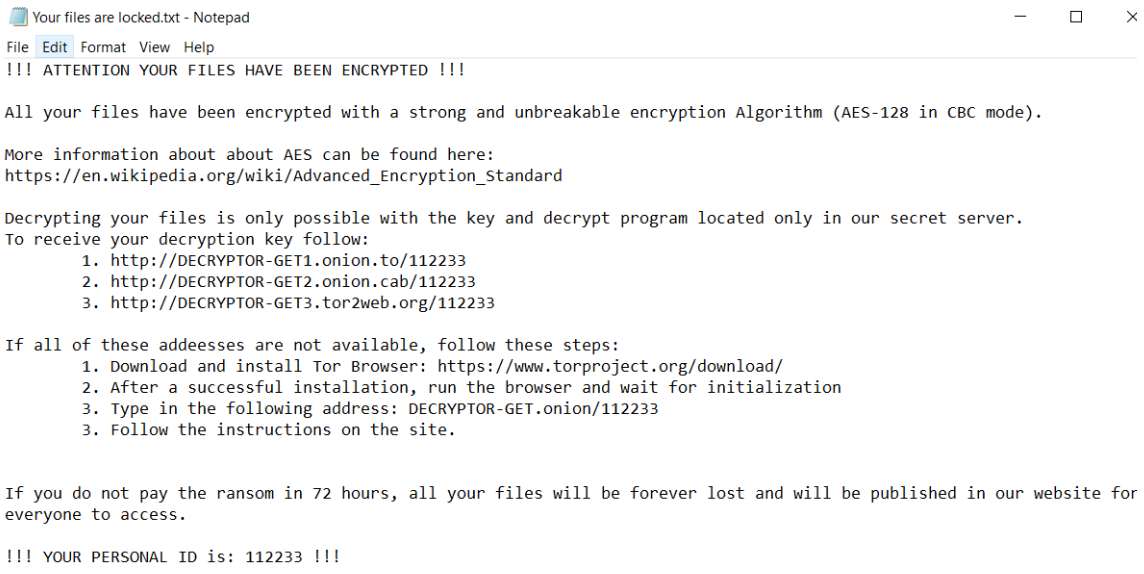
One may notice that the ransomware uses HTTP and not HTTPS. While HTTP offers no encryption, I assumed that all such traffic is encrypted, as this is a mechanism that attackers employ in genuine attacks. Therefore, instead of configuring the complex infrastructure of the exfiltration C&C servers, I simply assumed that all communications are encrypted and therefore packet inspection is inefficient.

Once all the files have been exfiltrated to the C&C server, the encryption of the files is triggered. To encrypt a file, the malware reads all the data of the file in byte format, uses the symmetric key it generated to encrypt them, overwrites all the data of the file with the encrypted data and finally closes the file.

After the encryption of the files finishes, the ransomware attempts to hinder the recovery capabilities of the victim. To simulate that step, it disables the Windows Recovery Environment by issuing the command “reagentc /disable”. Of course, this step could include a myriad of other techniques, such as searching for directories containing the word “backup”, or deleting shadow copies, but this method used here suffices to simulate that step of the attack.

Finally, after encrypting all the desired files, the ransomware drops the ransom note displayed in the figure below, informing the victim of the attack and the steps that they need to take to recover their files.

CONTROL FLOW DIAGRAM WILL BE ADDED EITHER HERE OR IN THE APPENDIX


Your files are locked.txt - Notepad
File Edit Format View Help

```
!!! ATTENTION YOUR FILES HAVE BEEN ENCRYPTED !!!

All your files have been encrypted with a strong and unbreakable encryption Algorithm (AES-128 in CBC mode).

More information about about AES can be found here:
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Decrypting your files is only possible with the key and decrypt program located only in our secret server.
To receive your decryption key follow:
        1. http://DECRYPTOR-GET1.onion.to/112233
        2. http://DECRYPTOR-GET2.onion.cab/112233
        3. http://DECRYPTOR-GET3.tor2web.org/112233

If all of these addeesses are not available, follow these steps:
        1. Download and install Tor Browser: https://www.torproject.org/download/
        2. After a successful installation, run the browser and wait for initialization
        3. Type in the following address: DECRYPTOR-GET.onion/112233
        3. Follow the instructions on the site.


If you do not pay the ransom in 72 hours, all your files will be forever lost and will be published in our website for
everyone to access.

!!! YOUR PERSONAL ID is: 112233 !!!
```

Figure 4-2: The Ransom Note dropped in the directories of the victim.

5 EXPERIMENTS

The main focus of this chapter is the description of setup, the experiments used to evaluate the created system and preparation that was necessary to perform those experiments.

I begin by describing the training dataset that was created, the data preprocessing and preparation steps that were necessary to train the Auto-Encoder, the training, and the tuning of the hyperparameters of the model and the threshold choices that were explored, as they all are an integral part of the experimentation phase. Subsequently, I describe the experimental environment of the test phase, what test data I am using and how the evaluation of the model will take place using that test data.

5.1 EXPERIMENTAL SETUP

To perform all the experiments, I initially needed to create the environment in which the detection model and the ransomware will be deployed.

The test device that was used during experimentations was a Windows Intel Core i9 personal workstation and was used to perform normal and daily activities. This device was used daily to perform normal and daily activities and was therefore monitored with the mentioned tools to create the training and test datasets. I personally used this device during the monitored period to generate legitimate traffic and have an experimental testbed for the developed system.

Furthermore, I created two simple servers in Python to use them as C&C servers. The first server listens on the port 3344 for connections and simply waits for two sets of messages being sent by the ransomware, the collected machine information, and the generated symmetric encryption key. The second configure server is the C&C server in charge of data exfiltration. It is a simple HTTP server listening on the alternative HTTP port 8080 and expects to receive HTTP POST requests by the ransomware containing the exfiltrated target files. As was said before, HTTP was chosen to try to blend ransomware traffic with normal. Also, it sufficed to use HTTP, instead of HTTPS, as it can simply be assumed that the packet's data are encrypted, as it is usually done by ransomware.

Additionally, to simulate the ransomware attack it was necessary to supply the data that will be exfiltrated and encrypted by the ransomware. To do that, I created a directory comprised of a vast plethora of subdirectories and files, including media, text files, documents, databases and more. It was important to simulate a realistic set of files so that the features extracted in the dataset would correspond to realistic file systems. Of course, these directories did not only include files that are targets for ransomware but include a variety of other legitimate files commonly found in any machine. During the attack, the ransomware traverses only these directories searching for files to exfiltrate and encrypt. The size of these files and directories is only 1.82 GB. However, as will be presented in the Results chapter, the system was able to detect ongoing attacks even when the ransomware targets such small quantities of files.

ADD FIGURE OF THE NETWORK TOPOLOGY/ARCHITECTURE OF THE SETUP CONTAINING THE TWO SERVERS, THE VICTIM AND THE RANSOMWARE RUNNING INSIDE THE VICTIM

5.2 Dataset Description

The final dataset that was used for the training of the detection system comprised of the features outlined in the Table 4. It consists of 33258 samples and 12 features that correspond to the various stages of ransomware attacks and were collected during one week of monitoring. During this week, the device was monitored for several hours to generate plenty and diverse activities.

As some processes may exhibit the same behaviors, it is expected that the dataset will include many duplicate entries. To correctly train neural networks, it is imperative to keep only one instance of all values, otherwise the model will be heavily influenced to learn those duplicate samples and overfit. After removing the duplicate values, the dataset is finally comprised of 3273 samples.

Next, the statistical properties of the dataset should be examined to ensure that it has an appropriate format. A statistical description of the dataset can be seen in the table below:

		mean	min	max
NUMERICAL	NET	6504.912007	0	358314
	FILES_WRITTEN	0.202871983	0	22
	AVG_WRITTEN_ENTROPY	4.150351026	1.584962501	7.998842444
	AVG_WRITES_FILES	0.858944794	0	190
	QUERY_DIRECTORY	26.71341277	0	6904
BINARY	WORKING_HOUR	0.503513596	0	1
	AUTORUN_EDITS	0.028719829	0	1
	EXISTS_IN_AUTORUN	0.003360831	0	1
	CRYPT_DLLS_LOADED	0.083409716	0	1
	ELEVATED	0.025358998	0	1
	KILL_DEFENSE	0	0	0
	DELETE_BACKUP	0	0	0

Table 5: Statistical Description of the training dataset.

Overall, the feature set comprises of both numerical and binary features. For this case, the range of each feature's values is the most important among the statistical properties of the numerical features. One can observe that the numerical features have different ranges, which also differ by orders of magnitude. As will be seen in the next section, that property can introduce instability and poor training performance to the model.

In terms of the binary features, the most important aspect to discuss is the balance of the classes. Primarily, between working and non-working hour samples there exists the required balance (as can be seen by the feature's mean), which introduces no imbalance and bias in the model during training. Hence, the Auto-Encoder is expected to train equally well for both samples of working and non-working hours.

In contrast, it is shown that the rest of the binary features are heavily skewed towards one category. For example, the "KILL_DEFENSE" feature does not include samples that tried to terminate defensive software. Nonetheless, that was due to the fact that this is an extremely rare behavior of normal applications and therefore there is no way to pursue a balance for these kinds of features, no matter how much the monitoring period is extended. In reality, as this behavior usually corresponds to malicious activities, the observation of such rare events will also partially influence the model to classifying them as anomalous, aligning with the goal of the developed system.

5.3 PREPROCESSING AND DATA PREPARATION

Before any training occurs, it is important to preprocess and transform the dataset into a format suitable for use by neural networks. To begin with, some of the features used are binary. Generally, it is considered best practice to use One-Hot encoding for binary features, so every binary feature in the dataset was transformed into two features in the following way:

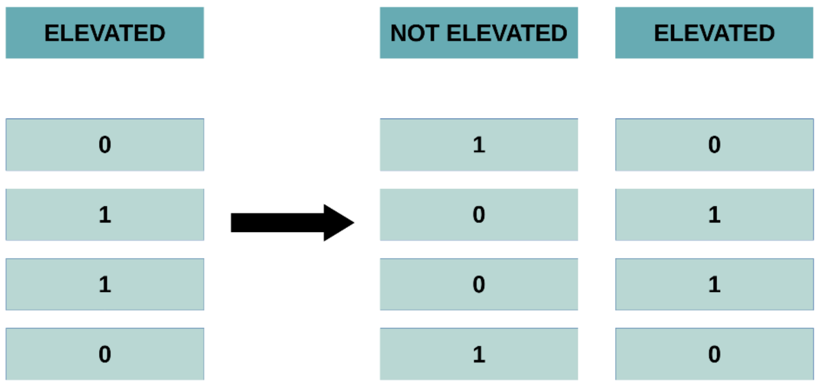


Figure 5-1: One-Hot Encoding of the Elevated feature as an example

Next, I discarded the duplicate values, and the dataset is split into two non-overlapping datasets, one used solely for training and the other one used for validation with a ratio of 90/10.

In the description of the dataset, it was mentioned that the numerical features have different scales. For example, the "NET" feature ranges from 0 to tens of thousands, while the "AVG_WRITTEN_ENTROPY" is bound between 0 and 8, by the definition of Shannon Entropy. Therefore, it is extremely beneficial to normalize or standardize the dataset, where normalization means that the resulting features will be casted to the range between [0,1] and standardization means that additionally the mean and standard deviation of each feature will be 0 and 1 respectively. During validation it was found that normalization works better for this case, so eventually each feature was transformed to have a range of 0 to 1.

What must be highlighted here is that only the training samples were normalized using its own statistics. To ensure that there is no data leaking to the validation and test datasets, both sets are normalized using the statistics of the training set. Therefore, while the training set is expected to have min and max values of 0 and 1, that is not true for the validation and test sets, although their scales are expected to not differ substantially.

5.4 Hyperparameter Tuning

A vital step of the experimentation phase is the choice of the right parameters of the model and other factors that influence its performance.

It is important to note that all choices that will be described here were made based on the performance of the model on the *validation* data only, and specifically the loss of the validation data corresponding to the normal behaviors observed. That is because all the test data (normal and ransomware data points) should not be used for any architectural decisions and should be used strictly for the final evaluation of the model's performance.

First and foremost, I experimented with different *loss functions*. As the outputs of the output layer of the Auto-Encoder are numerical values, I experimented with loss functions for regression tasks. Specifically, I experimented with two loss functions, the *Mean Squared Error (MSE)* and the *Mean Squared Logarithmic Error (MSLE)* the formulas of which are presented below:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Equation 4:Mean Square Error (MSE) function

$$MSLE = \frac{1}{N} \sum_{i=1}^{N} (log(1 + \hat{y}_i) - log(1 + y_i))^2$$

Equation 5:Mean Square Logarithmic Error (MSLE) function

A popular option to optimize models with is the MSE loss function. This involves minimizing the average of the squared errors or taking the mean of the squared differences between the predicted and actual values. However, while MSLE is not a popular choice, its utility depends on the actual use case. Among the advantages of the MSLE are that it utilizes logarithms to off-set large outliers in the dataset.

In this scenario, as much as we want to punish large deviations of the predicted outputs from the true values, we must do that frugally, as the essence of the model is to capture fine details in the differences between the normal and ransomware behaviors. As this model will be trained only on normal behavioral data that may contain some rare and "abnormal" but legitimate data points, these "outliers" should be treated also as normal.

Therefore, the normal outliers should not influence the model heavily and be incorporated into the model's perception of normal.

At the end, the Mean Squared Logarithmic Error was preferred, as it performed much better, providing more granular results and interpretations of the differences between the learned normal and abnormal ransomware behaviors.

Furthermore, it was essential to explore different *architectures* of the Auto-Encoder. Firstly, I analyzed the effects of different number of hidden layers of the Encoder and the Decoder. Specifically, I used Auto-Encoders of two, three and four hidden layers. Secondly, I experimented with different numbers of neurons in the hidden layers, including the number of nodes at the bottleneck. During this analysis, each subsequent layer had half the neurons of the previous layer, and the main choices that I tested were layers that would have the same number of neurons before they are reduced in the next layer. After extensive experimentation I uncovered that the best performing architecture is the one shown in the figure below:

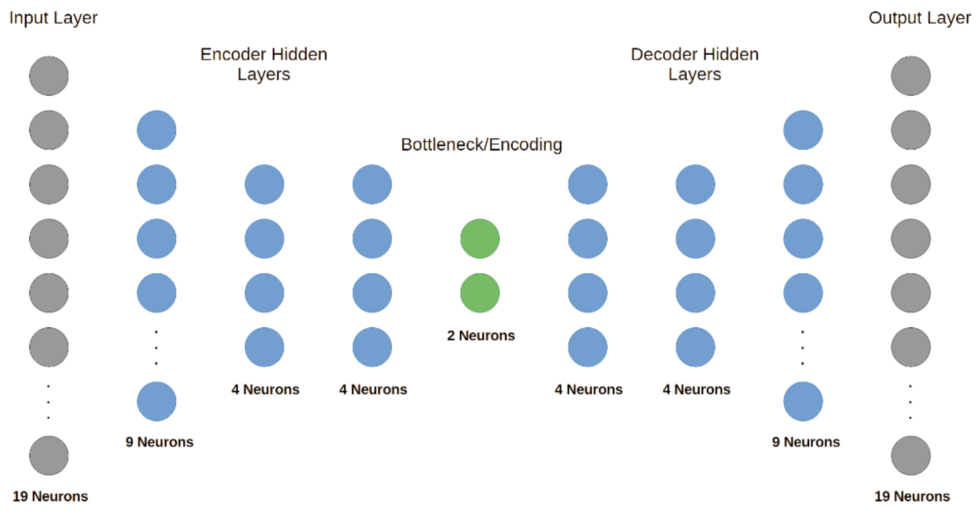


Figure 5-2: The Auto-Encoder's final architecture

The Encoder (and the Decoder in reverse order) overall has five layers, one input layer, three hidden layers and one bottleneck layer, the layer of the target latent representation. Its input layer has 19 input neurons, one for each of the input features used, accounting for the One-Hot transformations as well. The first hidden layer comprises of 9 neurons, which connect to the next hidden layer, which in turn has 4 neurons. The final hidden layer consists also of 4 nodes, keeping the number of neurons the same as the previous hidden layer. The bottleneck layer connected to the last hidden layer has two nodes, and thus the encoded representation of the data has only two dimensions.

The Decoder section is symmetric and therefore has as its input the bottleneck outputs and progresses in the reverse order to the Encoder until its output layer. Finally, the output layer of the Auto-Encoder has 19 neurons and therefore 19 output values which are fed to the loss function chosen to calculate the reconstruction error of the input.

All the neurons of the above layers except the output neurons use the Rectified Linear Unit (ReLU) activation function, as it was compared to the Tanh activation function and was uncovered that it performs slightly better. So, ReLU was preferred as an activation function additionally to the fact that it is faster than Tanh.

$$ReLU = max(0, x), \quad Tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Equation 6: ReLU and Tanh activation functions

Subsequently, I researched different *batch sizes,* that is the number of samples per gradient update. Generally, the batch size has a direct impact on the accuracy and computational efficiency of the training and evaluation process and can be seen as a trade-off between accuracy and speed. Smaller batch sizes allow the model to learn from each (or small sets) of examples but takes longer to train as the gradients are updated frequently. On the other hand, a larger batch size may lead to faster training time but may result in the model not capturing the nuances of the data. Thus, there isn't a one-size-fits-all answer for batch sizes. Instead, I explored different batch sizes, ranging from 8 to 64. Eventually the batch size that performed better on the evaluation data was 16 samples per batch.

In connection with the batch size, I explored the number of epochs that the model will be trained for, which can affect the generalization abilities of the model. An epoch is a single pass through the entire dataset and the number of epochs defines the number of times that the model will see the dataset. If the number of epochs is too small then the model may not learn the underlying patterns of the data, resulting in underfitting and if it is too large the model may overfit, leading to poor generalization performance on new, unseen data. After several runs, it was uncovered that the model training for 10 epochs performs greatly without suffering from overfitting or underfitting.

Moreover, the *optimizer* and the *learning rate*, that are used to minimize the loss and the step size taken, should be tuned as well. During this phase, I studied the Adam optimizer and the Stochastic Gradient Descent (SGD), using various learning rates. In general, Adam tends to converge faster, but sometimes suffers from generalization issues, while

SGD leads to slower training times, but converges to more optimal solutions. Although Adam is the most popular choice for model training, SGD's generalization capabilities make it still an interesting choice. As for this work I aim towards a detailed representation of normal behavior, which may have some fluctuations in their features, I chose to use SGD. The learning rate that was chosen 0.03 resulting in a good training speed without overshooting minima or resulting in underfitting.

Other hyperparameters that were explored were the *shuffling* of the dataset and existence of *Dropout* between the layers. Eventually, I chose to use shuffling, which shuffles the training and validation data during the training phase before each epoch, as it reduces overfitting of the network. Additionally, I chose not to implement Dropout, which is also used to reduce overfitting as my model didn't exhibit any such signs and therefore it was deemed unnecessary.

5.5 THRESHOLD CHOICES EXPLORATION

Perhaps one of the most important decisions that were made was the choice of a *threshold* for deciding when an event is classified as normal or anomalous. In general, any value that exceeds the threshold should be classified as an anomaly and triggering an alarm, and any value below that threshold should be conceived as normal activity. This way, the threshold directly influences the system's performance and affects the false negatives and false positives alerts generated, the two most crucial evaluation metrics of the system. For those reasons, the choice of the threshold should be made with special care, minimizing the number of false positives and false negatives, classifying each event at their true category.

Overall, I explored two main categories of thresholds, percentile-based thresholds, and Sigma rule-based thresholds.

A percentile is the value below which a percentage of data falls. Thus, if a score is said to be in the 90th percentile, this means that 90% of the scores in the distribution are equal to or lower than that score. Percentile-based thresholds define a threshold based on a defined percentile and thus classify observations as anomalies if they exceed the value of that percentile. Sigma rules are widely used heuristics for outlier detection of normally distributed data and a value is classified as an anomaly if it lies outside the mean by plus or minus "x" time sigma (σ), where σ is the standard deviation of the distribution and x is a positive integer. For instance, the most common sigma rule is the 3σ rule, which classifies as anomalies all observations that have a distance larger than three standard deviations from the mean of the distribution.

Of the two common choices for thresholds, I deemed early that percentile-based thresholds are not a prudent choice, as they assume that anomalies exist but are rare in the training data. This though does not hold for this case, as the model is trained exclusively with normal data and by definition it would generate false positive alerts on normal data. Therefore, sigma rule-based thresholds are more suitable for this work.

Originally, sigma rules are used for data that are normally distributed and one might think that they are not appropriate for this case. Nevertheless, I advocate the contrary. Firstly, the number of standard deviations can be tweaked and be manually chosen to minimize the number of false positives generated encapsulating the whole range of the distribution for large enough standard deviation numbers, something that is not true for percentile-based thresholds. Secondly, even though the reconstruction losses are not expected to be normally distributed, most of the losses will concentrate on the low values of the distribution and very few samples will result in high loss, primarily those belonging to ransomware behaviors. Therefore, normal data may not exist above a chosen standard deviation, leading to minimal or even no false negatives being generated. For those reasons, a proper choice of a standard deviation from which the sigma rule threshold will be calculated may result in minimal to no false negatives.

During experimentation, different percentiles and standard deviations were investigated to choose a suitable threshold. Specifically, three percentiles and four standard deviations were explored: the 90th, 95th and 99th percentiles and the 3σ, 5σ, 7σ and 10σ. From the obtained results the previously described intuition was verified, and the sigma rules outperformed the percentile-based rules. Specifically, the best performing sigma rule was the 5σ rule which produced no false negatives. Of course, larger standard deviations also produce no false negatives. Five standard deviations were chosen as the most suitable threshold choice, as supplementarily to the fact that it generated no false negatives, it is not much larger than the largest reconstruction losses of the training data, enabling the detection of ransomware events that have relatively low reconstruction losses. This way, this threshold is expected to both correctly classify normal events as normal as they all will lie under that threshold and, it is also expected to detect ransomware events that are not deviating significantly from normal behavior, corresponding to the earliest stages of ransomware attacks.

5.6 Training

Of course, the training of the model is an iterative process that included the steps described above, such as hyperparameter tuning and choosing an appropriate threshold. In this section I describe the last training round which resulted in the final model that is used for the early detection of ransomware attacks. Having finalized the structure and contents of the datasets, the architecture of the model and its hyperparameters it's then time to train it for the final time.

The model was overall trained for only 3.5 seconds, due to its very small size (517 parameters and 2.26 MB). In the figure below, it is seen that the model was trained for a total of 10 epochs, during which the training and validation errors decreased and converged to 0.0198 and 0.02 values respectively. Additionally, the training errors did not experience any sudden changes and followed an overall smoothly declining path, validating the correct choice of the loss function, the optimizer, and the learning rate.

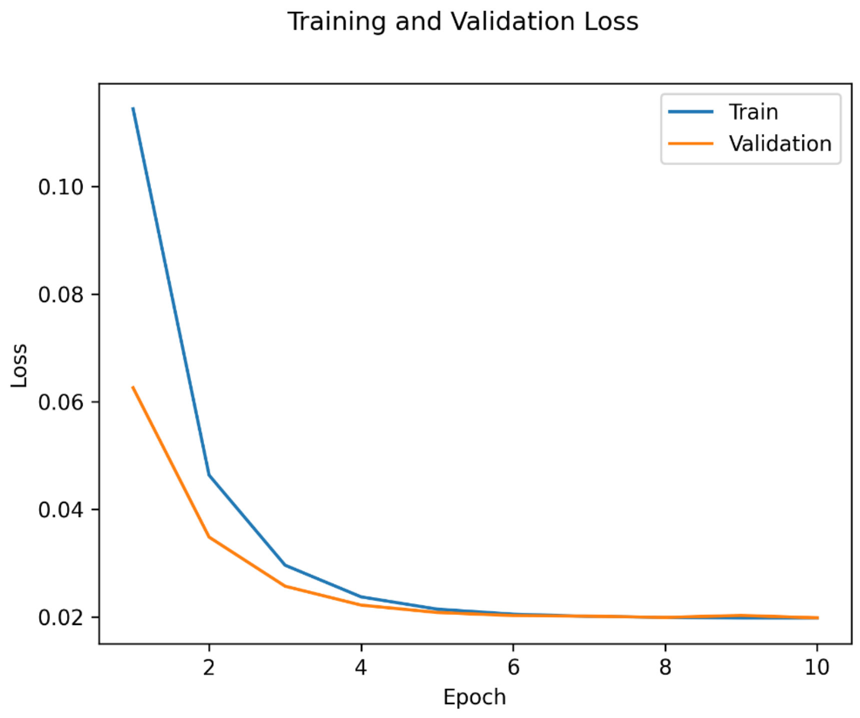


Figure 5-3: Training and Validation Loss Curves

It is apparent that the validation and loss curves have a very small distance between them, signifying that the model does not show any signs of underfitting or overfitting. In fact, since that difference is that small, the model is expected to capture the complex characteristics of the training data and to generalize and perform equally well in the unseen data of the testing phase. This behavior is very encouraging as the desired performance of the system is to be able to perform well on the new test data.

Furthermore, from the training data's reconstruction errors, the threshold value that was calculated was 0.1095. As a reminder, any value above that value will be classified as an anomaly and all the values below that will be considered normal.

After training the Auto-Encoder for the final time and validating that it exhibits sufficient properties to be tested, it should be used on test data to explore its efficiency in detecting ransomware attacks in their earliest pre-encryption stages.

5.7 Proposed Experiments

As mentioned previously, the system's capabilities should be explored with a rigorous set of experiments. Throughout all the experiments that will be described below, the user is expected to interact with the test device for some minutes to generate legitimate traffic according to the time context: working and non-working hours. During this time, the device will be monitored using procmon and the developed PowerShell script to extract the features as was explained in the previous chapters. At some point, the test ransomware will be executed on the device to simulate an attack. After the attack is concluded, the test dataset created will be fed to the detection model, the Auto-Encoder, to attempt to detect any anomalous activities.

When the Auto-Encoder generates its output, it will be obvious which events were classified as normal or anomalous, due to the chosen threshold value and at which stage the attack was detected. Additionally, the Confusion Matrix of the monitored period's events can be constructed to evaluate the model's performance. This is possible, as procmon allows us to identify the ransomware process from its Process ID. Using the ransomware's Process ID, the test dataset can be split into normal and ransomware events, enabling us to create the Confusion Matrix.

STATE THAT IN THE FIRST EXPERIMENTS THE RANSOMWARE IS BASED ON A SINGLE PROCESS

The first set of experiments performed concerns the detection capabilities of the model when an attack is performed during *working hours*. When the test device is used to conduct the legitimate activities of the user during working hours, a large volume of diverse activities is anticipated. In fact, many of the activities may produce behaviors that may be similar to some aspects of an attack. For example, during working hours, a user may edit many files, may send data and files over the web, may use privileged processes to perform some specific actions and more. However, as these normal processes are not expected to exhibit many of the characteristics of an attack simultaneously, the system should be able to discern normal activity from ransomware with a high level of accuracy.

A few tolerable false negatives may be observed, which may correspond to the very first steps of the ransomware, before the ransomware process has done anything significant.

For the second set of experiments, the model should be tested during non-working hours. During this experiment, the user is expected to interact notably less with the device than in working hours, both in terms of volume and set of actions. For example, the user is expected to modify less files and with a lower frequency, less processes that may edit the registry to be active, to traverse smaller parts of the file system and to send less data to other hosts. For that reason, differences between normal and ransomware behaviors will be larger than in the first experiment and it is expected that the detection system will be able to correctly detect anomalous events that maybe were not that apparent previously.

Furthermore, it is beneficial to explore the capabilities of the model when the ransomware tries to use some techniques to evade the defenses of the system. Unquestionably, attackers may use a plethora of ways to evade any defense mechanisms employed. However, in these experiments I use a small set of techniques that are sufficient to explore the effectiveness of the model across different environments. Additionally, the set of experiments that are described below should be performed during both working and non-working hours and complement their results. All the experiments except one correspond to the use of other processes to perform certain tasks of the attack.

In the previous chapters, I outlined that the attackers may statically link their code to encrypt files. To simulate that, I need only to modify the "CRYPT_DLLS_LOADED" feature in the ransomware samples and change them all to 0, indicating that that process did not use any encryption APIs.

Additionally, attackers may use other processes to conduct certain aspects of the attack. For this subset of experiments, I explored the cases where ransomware spawn new processes to perform two tasks: terminate processes that relate to defensive mechanisms of the system and modify the registry keys to gain persistence. To explore their effects, I modified the ransomware code to spawn other processes and perform these tasks.

During experimentation, I performed these sub-experiments in an incremental fashion. To illustrate, the first sub-experiment simulates the static linking of encryption libraries. The second sub-experiment spawns a process to terminate the defensive software additionally to the statical linkage of the libraries. Finally, the third sub-experiment spawns another process to modify the registry keys.

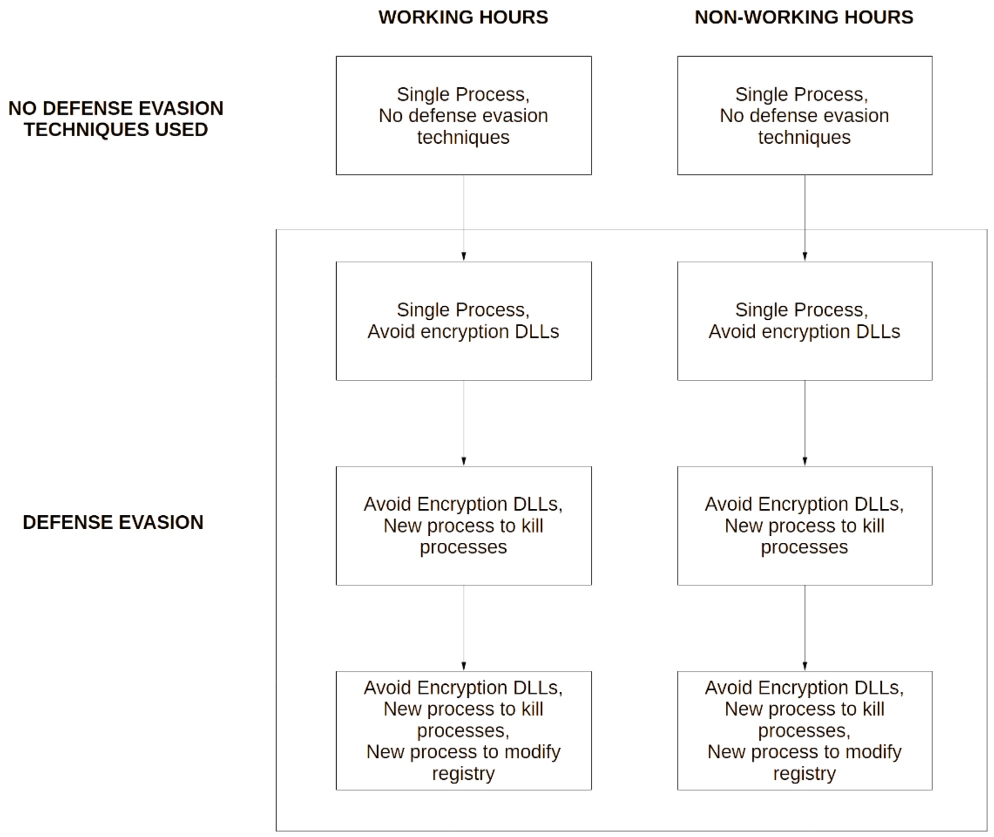

WORKING HOURS
NON-WORKING HOURS
NO DEFENSE EVASION TECHNIQUES USED
Single Process, No defense evasion techniques
Single Process, No defense evasion techniques
DEFENSE EVASION
Single Process, Avoid encryption DLLs
Single Process, Avoid encryption DLLs
Avoid Encryption DLLs, New process to kill processes
Avoid Encryption DLLs, New process to kill processes
Avoid Encryption DLLs, New process to kill processes, New process to modify registry
Avoid Encryption DLLs, New process to kill processes, New process to modify registry

Figure 5-4: The set of experiments conducted.

After describing the experiments that were needed to evaluate the performance of the model, the next section presents their results.

6 RESULTS AND DISCUSSION

In this section I describe and discuss the results of the experiments proposed in the previous chapter. In the beginning, I present the results of the system when the test ransomware employes no defense evasion techniques, and later I show the performance of the detection system when the malware attempts to evade defense mechanisms.

As was noted in the previous sections, the main output of the detection system is the reconstruction errors of the Auto-Encoder's outputs, which is then compared to the threshold value. For all experiments I used the device for some minutes while it was being monitored and deployed the ransomware so the attacks can be performed. After the attacks were concluded, I formed the respective test datasets, preprocessed them as was described in section 5.3 and I fed them as input to the Auto-Encoder to detect any anomalies. Finally, I plot the reconstruction errors derived from the Auto-Encoder.

To easily discern any false positives and false negatives generated, in each test experiment I separate the samples of the ransomware process from the rest normal processes. The two sets of samples are still plotted on the same figure but are separated with a vertical red line. The samples of normal processes are shown on the left side of the red divider and the ransomware reconstruction errors are shown on the right side of the red line. Finally, in each figure a vertical black line is shown that indicates the threshold value. Thus, any point that is above the threshold is classified as an anomaly and the points below the horizontal line are classified as normal.

Before presenting each experiment's results it is necessary to describe a trend observed in all scenarios. In all cases, the reconstruction loss follows a similar pattern to the one seen in the diagram below. This diagram does not correspond to any specific experiment but just presents the trends of the ransomware reconstruction errors.

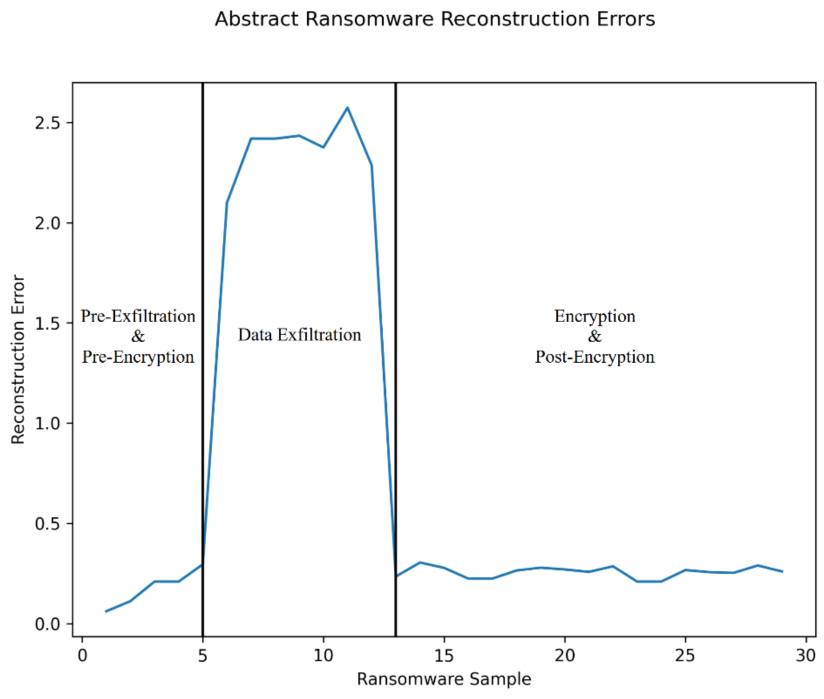


Figure 6-1: Abstract behavior of the ransomware reconstruction errors.

To understand the results of the conducted experiments it is important to interpret that pattern. By examining this general plot, three distinct areas can be observed. Initially, the reconstruction errors of the ransomware samples start from their lowest points and gradually increase, until they shoot up abruptly. These samples correspond to the stages where the ransomware escalates its privileges, loads Windows encryption DLLs, terminates other processes, modifies the registry to gain persistence and finally traverses the directories of the system in search of files to encrypt. This continuous growth of the errors occurs because as the ransomware advance through their kill chain and perform increasingly more malicious actions, the combination of those activities can also increasingly describe an attack. This behavior is aligned with the hypothesis that the combination of the features can be used to correctly discern normal from anomalous behavior and provides a promising initial result for the detection of attacks at their early stages. Therefore, these samples generally correspond to the Pre-Exfiltration & Pre-Encryption phase. Subsequently, there is a massive surge of the reconstruction errors for several samples, constituting the second distinct phase. These high reconstruction errors appear because the ransomware exfiltrates a very large amount of data to the C&C server, which in combination with the rest of the features, renders these behaviors profoundly different from the normal baseline. When the data exfiltration finishes, the reconstruction errors drop again and stay relatively stable, fluctuating slightly until the end of the ransomware process's life. This third and last period corresponds to the Encryption and Post-Encryption stages of the ransomware.

Undoubtedly, one shouldn't be discouraged when the reconstruction errors are low in comparison to the large peak of the exfiltration phase, as the level of these errors is meaningful only when compared to the chosen threshold. As will be shortly seen, the detection system succeeds in detecting the ransomware in the Pre-Exfiltration & Pre-Encryption phase in all the experiments, not allowing the ransomware to approach even the exfiltration phase. The only difference of each experiment is the exact stage of that phase where the system managed to capture the attack.

6.1 WORKING HOUR RESULTS

The first set of experiments conducted was the series of ransomware attacks during the working hours of the day.

6.1.1 NO DEFENSE EVASION EMPLOYED

In this experiment, the ransomware that conducts all the activities in a single process was deployed. After the creation of the test dataset, the normal test samples included 867 samples, and the ransomware 29 samples.

After the dataset was fed to the model, the Auto-Encoder produced the following result:

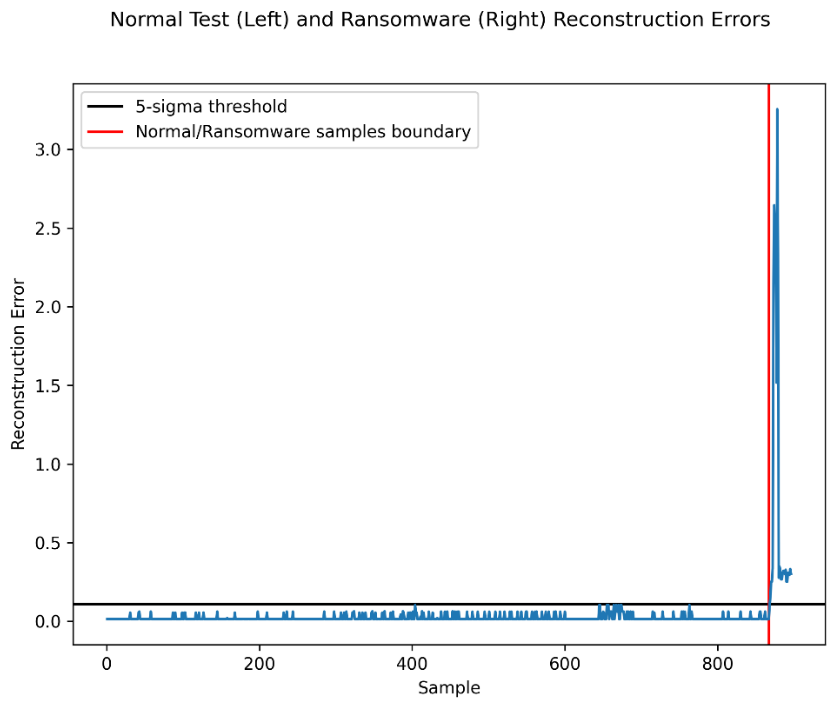


Figure 6-2: Result of the ransomware being deployed during working hours and employing no defense evasion techniques.

From the figure it is seen that *all* the normal processes' behaviors are correctly classified as normal, generating no False Positives (0% False Positive Rate). Additionally, it is seen that the ransomware observations result in a large and continuous spike of reconstruction errors, validating the fact that the Auto-Encoder models the normal processes only and fails to accurately model any behaviors that do not conform to the learned legitimate patterns.

Furthermore, all the ransomware samples except one are correctly detected as anomalies, which has a reconstruction error of 0.103, resulting in only 3.45% False Negative Rate. Upon inspection of the ransomware samples, it was discovered that this sample corresponded to the first second of the execution, when the ransomware had just escalated its privileges and loaded cryptographic DLLs, which could also be exhibited by normal processes. As up until this point the malware did not exhibit any seriously malicious activities, the model didn't classify it as an anomaly. However, once the ransomware attempts to terminate any processes, the model correctly classifies that and all the subsequent observations as anomalies.

This behavior is radical, as for this scenario the model can successfully detect an attack before any user file has been encrypted or even exfiltrated! It can capture an attack in its earliest stages, specifically, once the ransomware has escalated its privileges, has loaded any cryptographic DLLs, and attempts to gain persistence.

Since the detection system performs so outstandingly in this scenario where the ransomware uses a single process for all the actions, it is important to test the system with more sophisticated ransomware, that employ defense evasion techniques and record the system's performance.

6.1.2 Experimentation with Defense Evasion Techniques

As the ransomware gets more sophisticated and tries to evade the defenses of the device the reconstruction loss of all the ransomware samples tends to drop and the ransomware gets detected later in comparison to the above experiment. However, as will be explained shortly, the model is still able to detect the attack before encryption has occurred.

In the first sub-experiment, the ransomware avoids the use of Windows Encryption APIs. To simulate this, it sufficed to just change all the entries of the ransomware samples of the "CRYPT_DLLS_LOADED" feature to 0 and feed the same dataset to the Auto-Encoder to detect any anomalies. The reconstruction losses of that sub-experiment are seen in the figure below:

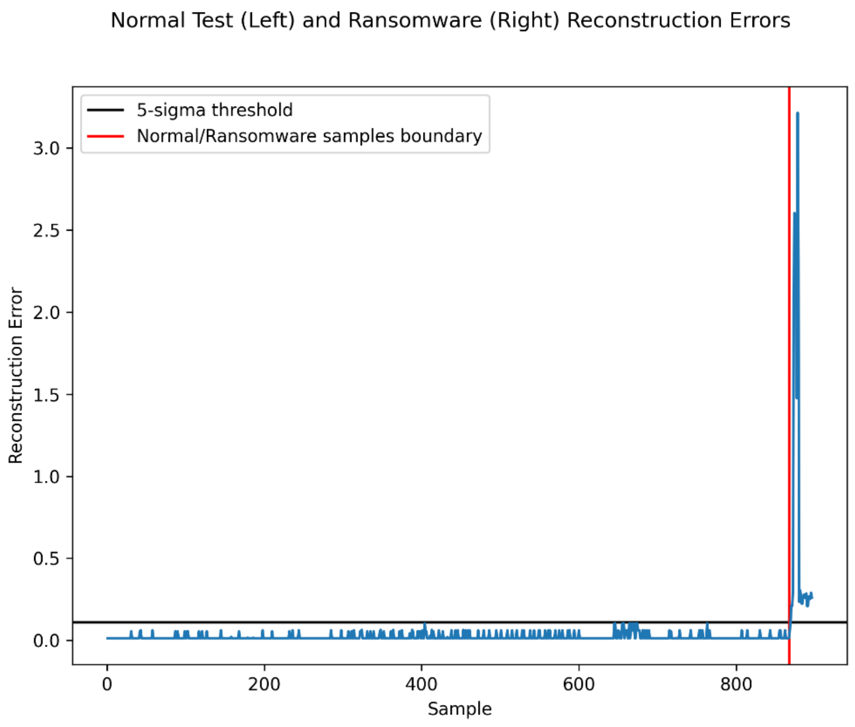


Figure 6-3: Result of the ransomware being deployed during working hours and avoiding Windows Encryption APIs

Overall, the detector exhibits a similar behavior, producing no False Positives for the normal processes. It fails to detect only the first observations of the attack, when the ransomware has only escalated its privileges and has not yet terminated any processes. The model then succeeds to detect the attack once the ransomware tries to kill any possibly inhibiting processes. However, it seems that the reconstruction errors for this case have slightly declined and approached the threshold, indicating that additional evasion techniques may allow the ransomware to escape the detection system.

Therefore, the model captures the attack, before any asset is exfiltrated or encrypted, specifically once the ransomware attempts to kill any processes that might hinder the rest of the attack and continues to detect all anomalies until the end of the attack.

In the next sub-experiment, the ransomware additionally spawns a new process to kill any processes that might inhibit the rest of the attack. The normal test set is comprised of 439 samples and the ransomware set of 29 samples.

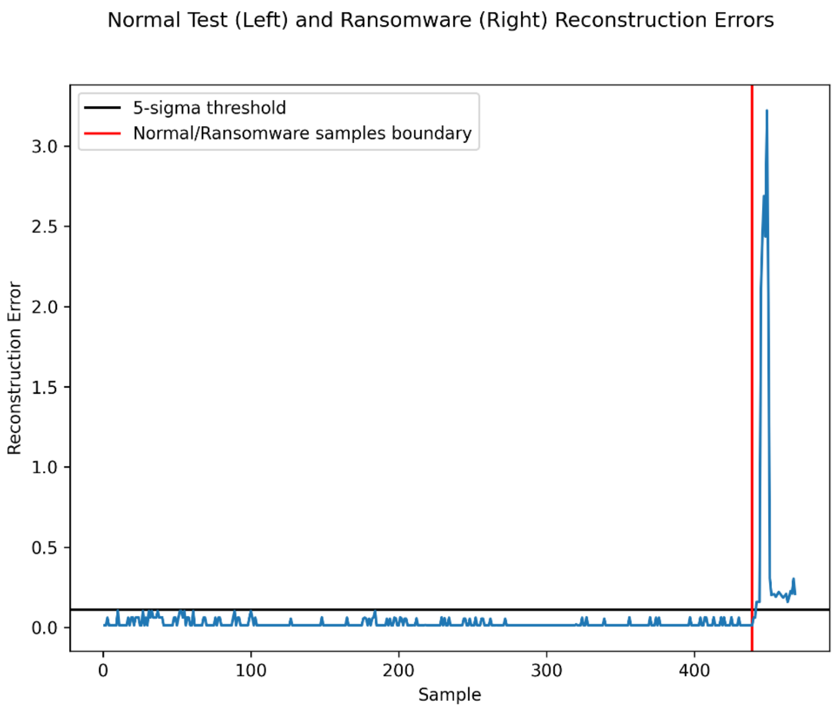


Figure 6-4: Result of the ransomware being deployed during working hours, avoiding Windows Encryption APIs and spawning a new process to kill any defense mechanisms.

In this scenario, the system first correctly classifies normal behaviors as normal resulting with a 0% False Positive Rate once more. However, the efficiency of its anomaly detection capabilities seems to drop, as the reconstruction errors of all the ransomware samples have significantly dropped in comparison with the previous cases. Specifically, in Figure 6-4 above it is shown that the first two malicious samples are incorrectly perceived to be normal behaviors, resulting in a 6.89% False Negative Rate. This way, the ransomware manages to escalate its privileges, and kill processes without being detected as an attack.

Nevertheless, the system detects the attack once the ransomware attempts to interact with the AutoRun keys and all subsequent observations of the same process are correctly classified as anomalies.

As mentioned in the dataset creation section, the feature "EXISTS_IN_AUTORUN" exists to capture the scenario where the ransomware may use other processes to modify the registry. Although the main ransomware code spawns another process to modify the registry, this feature manages to capture that attempt. It therefore triggers the detection system to raise an alarm and detect the attack once the ransomware attempts to gain persistence and most importantly before it accesses the user's files.

Despite that small drop in performance, the system manages once more to capture the attack in its early stages before any data exfiltration or encryption has occurred. It remains to see now how the model will react if the ransomware utilizes one more defense evasion technique.

In the final sub-experiment of this set, the ransomware spawns yet another process to modify the registry, additionally to the one that was used to terminate the defensive processes, and to the fact that it avoids to use Encryption APIs.

For this experiment, the test dataset included 481 normal samples and 29 ransomware samples. After the dataset is fed to the Auto-Encoder, the following output is produced:

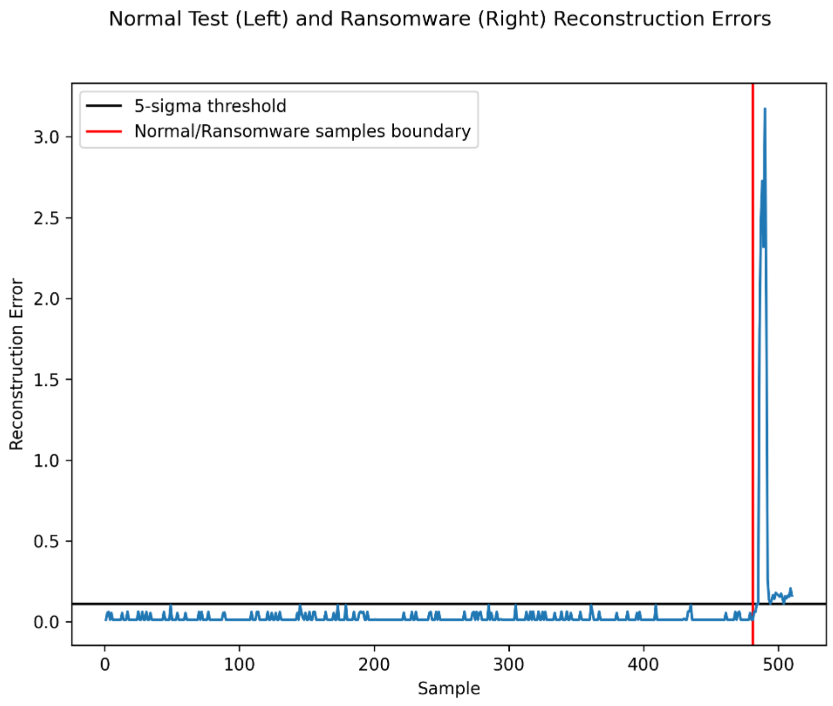


Figure 6-5: Result of the ransomware being deployed during working hours, avoiding Windows Encryption APIs, spawning new processes to kill any defense mechanisms and modify the registry

In this case, the results are almost identical to the previous experiment. The model manages to correctly classify all the normal samples correctly, resulting in 0% FPR and thus raising no alarms for any normal process.

Concerning the attack, the system fails to detect it again for the first two seconds, which correspond to the stages where the ransomware escalates its privileges and kills its target processes. Then, similarly to the previous scenario, the attack is detected once the ransomware is added as an AutoRun program, even though the child process interacts with the registry. As mentioned before, although this process does not itself modify the registry, the feature "EXISTS_IN_AUTORUN", enables the detection of the attack even in such cases.

Additionally, it can be seen that the reconstruction losses of the ransomware process have decreased again, approaching the threshold even more. Although most errors remain clearly above the threshold, some values exceed it by a small degree. This means that if more sophisticated methods are employed by the ransomware, the system may fail to detect them as anomalies.

Nevertheless, in this setting the detection system succeeds once more in the detection of the attack in its pre-encryption stages, which is the primary focus and contribution of this work.

6.2 NON-WORKING HOUR RESULTS

In the upcoming experiments, the attacks were conducted during non-working hours. The sequence of experiments is similar to the one in the previous section.

6.2.1 NO DEFENSE EVASION EMPLOYED

In the first main experiment, no defense evasion techniques are employed by the ransomware, and it is deployed during non-working hours. In the figure below, the reconstruction errors generated from the Auto-Encoder are shown:

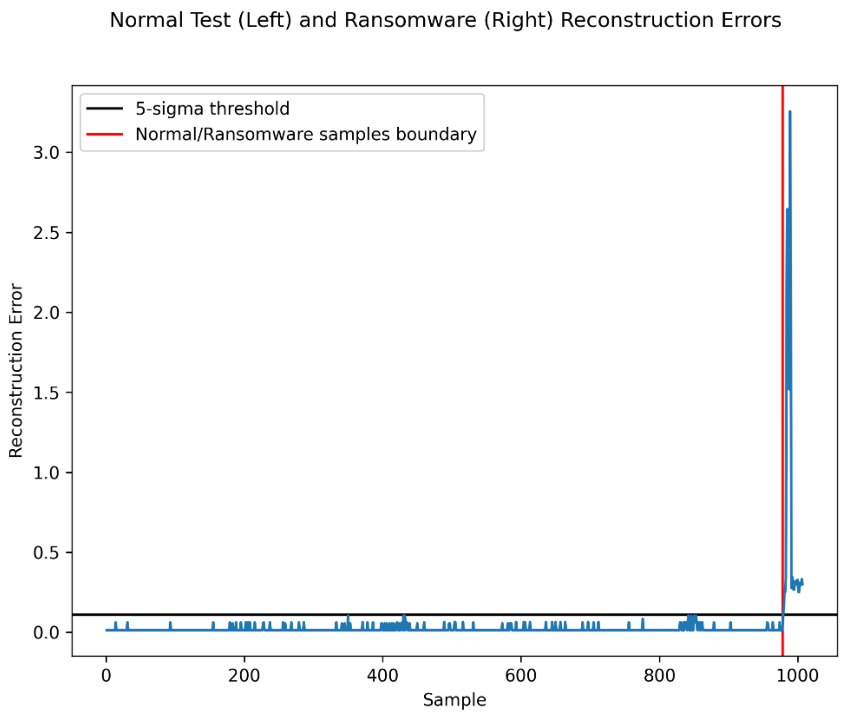


Figure 6-6: Result of the ransomware being deployed during non-working hours and employing no defense evasion techniques.

Similarly to the corresponding experiment of the working hours, *all* the normal processes' behaviors are correctly classified as normal, resulting in 0% False Positive Rate. Once more, the ransomware samples exhibit an abrupt and continuous spike, which is the intended behavior of the model and only one observation is classified as normal, exactly as in the case of the very first experiment. Here, the attack is captured once it attempts to kill any process of the system and continues to detect all subsequent behaviors of the process as anomalous, resulting in a 3.45% False Negative Rate.

It is important to highlight that the ransomware reconstruction errors during the non-working hours are larger than the errors of the ransomware of the working hours. Although the difference is small, this behavior is anticipated as normal operations in non-working hours are different and less intense than the ones observed during working hours.

Therefore, in this scenario as well, the system can detect ransomware attacks before they encrypt or exfiltrate the victim's assets with a high degree of confidence.

6.2.2 Experimentation With Defense Evasion Techniques

In the first sub-experiment using evasion techniques, the ransomware does not make any Windows Encryption API calls and this behavior is again simulated by running the previous experiment but changing only the ransomware "CRYPT_DLLS_LOADED" entries to 0:

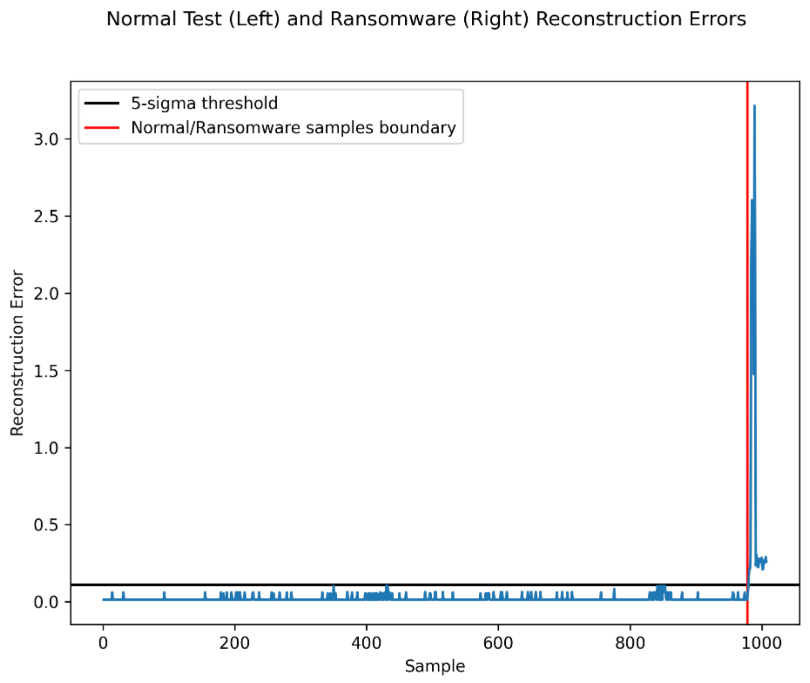


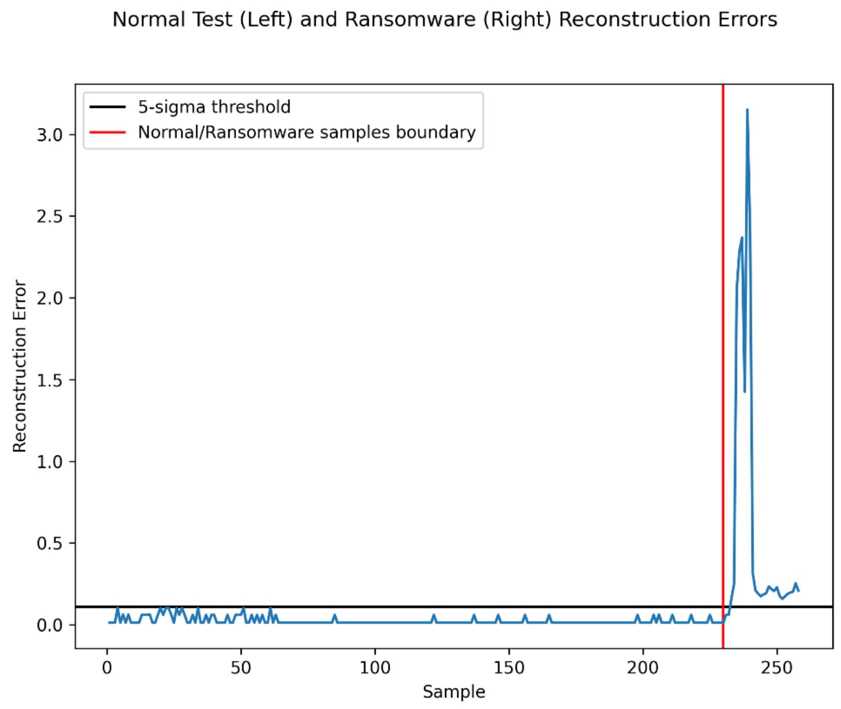
Figure 6-7: Result of the ransomware being deployed during non-working hours and avoiding Windows Encryption APIs

In this sub-experiment as well, all normal samples are correctly classified as normal leading to a 0% False Positive Rate. Like the previous experiments, the ransomware samples trigger a sudden rise in the reconstruction errors, indicating the awareness of anomalous incidents, but fails to capture the attack only during its first second of execution (3.45% FNR).

This experiment as well showcases the ability of the system to detect an ongoing attack early in its kill chain before any asset of the victim has been encrypted or exfiltrated.

In the second sub-experiment of this set, the ransomware spawns a new process that terminates various defensive processes, additionally to the fact that it doesn't make any Windows Encryption API calls.

During this experiment there are 230 normal samples and 28 ransomware samples. After conducting the attack, the Auto-Encoder produces the following reconstruction errors:



`

Figure 6-8: Result of the ransomware being deployed during non-working hours and avoiding Windows Encryption APIs and spawning a new process to kill any defense mechanisms.

During this experiment, the detector didn't produce any False Positives as well, but the efficiency of the detection system dropped, as can be observed from the diagram. By employing more defense evasion techniques, the ransomware managed to stay undetected up until the point that it attempted to modify the registry to get persistence, misclassifying the first two seconds as normal (6.89% FNR). Additionally, it is observed that there is a general drop in the reconstruction loss of the ransomware samples because the tail of the plot is approaching the threshold, signifying that more evasion techniques employed reduce the performance of the model.

Despite that drop in efficiency, the detector once again succeeded in the early detection of the attack and can alert the security analysts before the assets of the victims are harmed.

In the final sub-experiment, the ransomware employed one more defense evasion technique. Specifically, it spawns a second process to add itself as an AutoRun program and gain persistence.

This experiment is comprised of 375 normal samples and 29 ransomware samples. Their reconstruction losses are shown in the picture below:

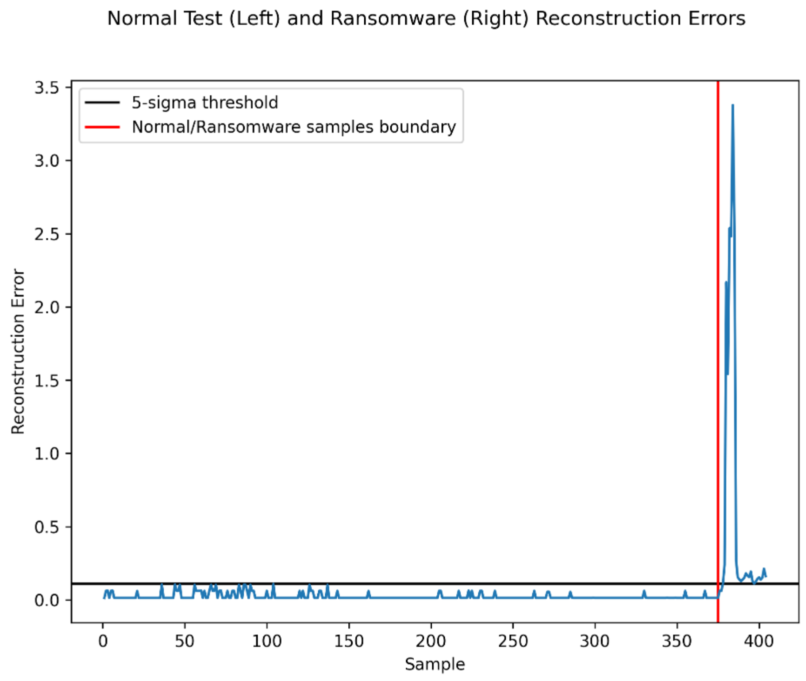


Figure 6-9: Result of the ransomware being deployed during non-working hours, avoiding Windows Encryption APIs and spawning new processes to kill any defense mechanisms and modify the registry

Following the pattern of the previous experiments, this case does not generate any False Positives, resulting in a 0% FPR. However, similarly to the corresponding case of the working hours experiment, the system manages to detect the attack when the ransomware attempts to edit the registry keys, allowing the ransomware to escalate its privileges and additionally kill any unwanted processes.

On the other side, the system is observed to lose more efficiency as the tail of its reconstruction loss seems to have shrunk and approached the threshold even more. While the system still classifies the rest of the ransomware behaviors correctly as anomalies it seems that additional defense evasion techniques employed by the ransomware may enable it to perform some of its activities unnoticed.

Overall, the Auto-Encoder results in a low reconstruction loss for all normal behaviors and thus it correctly classifies all normal behaviors as normal, resulting in a 0% False Positive Rate for all experiments conducted. Additionally, the system succeeds in detecting the various attacks early, before any encryption or exfiltration has occurred. In most experiments, the attacks are captured once the ransomware attempts to terminate any unwanted processes. However, as ransomware become more sophisticated, the system fails to detect that activity and instead detects the attacks once the AutoRun registry keys are interacted with. Despite that fact, the system detects the attacks before any files are encrypted or even exfiltrated.

Both these results are crucial as they show that with the correct feature set and a robust anomaly detection system, ransomware attacks can be captured in the very early and pre-encryption stages, allowing the mitigation of the risks of an ongoing attack.

In the following two tables, the summary of the results of the experiments is shown. Specifically, three important pieces of information are shown for each experiment. First the False Positive Rate, which is 0% for all scenarios, the False Negative Rate and finally the pre-encryption stage where the attack was detected by the developed detection system:

	WORKING HOUR		
	FPR	**FNR (# samples)**	**Stage at which the attack was detected**
- Single Process - No Defense Evasion	0%	3.45% (1 sample)	Defense process termination
- Single Process - Avoid Encryption DLLs	0%	3.45% (1 sample)	Defense process termination
- Avoid Encryption DLLs - New process to kill processes	0%	6.89% (2 samples)	Registry interaction
- Avoid Encryption DLLs - New process to kill processes - New process to modify registry	0%	6.89% (2 samples)	Registry interaction

Table 6: Summary of experiments results for the working hour experiments.

	NON-WORKING HOUR		
	FPR	**FNR (# samples)**	**Stage at which the attack was detected**
- Single Process - No Defense Evasion	0%	3.45% (1 sample)	Defense process termination
- Single Process - Avoid Encryption DLLs	0%	3.45% (1 sample)	Defense process termination
- Avoid Encryption DLLs - New process to kill processes	0%	6.89% (2 samples)	Registry interaction
- Avoid Encryption DLLs - New process to kill processes - New process to modify registry	0%	6.89% (2 samples)	Registry interaction

Table 7: Summary of the experimental results of the non-working hour experiments.

7 LIMITATIONS AND FUTURE WORK

Of course, the developed system is not a panacea, but has certain limitations and drawbacks. For this work to be complete, it is necessary to present its limitations and explain ways that these weaknesses can be remediated in future works. Thus, in this chapter, I first describe this work's shortcomings and I subsequently present possible paths succeeding research can take to remediate them and improve the general system.

7.1 LIMITATIONS

The primary limitation of this work revolves around the fact that the system developed is focused on ransomware that leverage a single process to conduct the attack. As was presented in the previous chapters, the test ransomware relies on a single process to perform all the steps of the attack, from privilege escalation to the drop of the ransom note. This though, might not reflect ransomware that are commonly met in the real world. By focusing on standalone ransomware, this work intended to firstly explore the possibility of timely detecting attacks in their pre-encryption stages via a combination of diverse features extracted from the various attack stages and machine learning algorithms, and secondly, to form a solid foundation for further works and research on that and similar topics.

In connection to the above constraint, it is unknown how the developed system performs on ransomware that are used in real attacks. Since this system was evaluated using a custom ransomware, its detection capabilities against legitimate attacks are not known. Modern ransomware usually deploy a myriad of defense evasion techniques and rely on complex mechanisms to become stealthy, and as was seen during the experiments, the inclusion of certain defense evasion techniques in the attacks, reduces the performance of the model in capturing them. Although the developed system demonstrated the capability of the early detection of simple ransomware attacks, this work does not address realistic ransomware that are met in the wild and attempt to conduct the attack in a stealthy fashion.

Additionally, one important constraint of this work is that the system was trained using a single person's activities. During the creation of the datasets, a single user was monitored and his activities were recorded over time. Although after training the model had learned how to accurately describe and encapsulate the patterns of the normal behaviors of an operator, these behavioral patterns may differ significantly from other users. This introduces a significant bias in the system, and it may be less efficient when deployed on other users' devices whose behaviors are dissimilar to the initial subject. Similarly, the model potentially holds prejudice against other devices as well, that may have previously unseen applications running. To illustrate, if the model was deployed on another device after training, even if the same user used that device, there is a possibility that that device's applications may significantly differ from the ones that ran in the monitored device, leading the system to generate many false positives. Of course, this issue concerns itself primarily with the risk of raising false alarms and not with failing to detect an attack. This is because a normal operation on a different device may seem unusual to the trained model, while ransomware activity will still seem anomalous.

Last but not least, the feature set used does not encapsulate all the possible ways that an attack may perform certain steps. For example, in features such as the "KILL_DEFENSE", an assumption was made that the ransomware would use "taskkill" to terminate other processes. However, attackers may use a plethora of other mechanisms to disable threatening processes that are not taken into account by the developed system. Although the consideration of some of the used methods suffices for the purposes of this work, the ignorance of other methods reduces its expressive and detection capabilities.

7.2 FUTURE WORK

One additional endeavor of this research is to form the basis for future work that will help improve the developed system, remediate its limitations that were proposed above and overall progress the early detection of ransomware attacks. Specifically though, there are certain paths that could be followed to extend and ameliorate this system.

First and foremost, it will be fruitful for both the research community and the industry to take into account ransomware that use multiple processes throughout the attacks. One possible way to do that is to use the current feature set as a basis and create a new one that offers a more high-level view of the system as a whole, in place of relying on a per-process-driven detection method. This new system could leverage the utility of some of the features crafted here and create new ones that provide a bird's-eye view of the entire system and its behavior as one entity, instead of detecting certain processes that might exhibit malicious behaviors. Finally, this system will be able to detect ransomware not from a certain process's behavior, but from the whole device's behavior.

Furthermore, as was stressed before, it is very important to test the detection system against real ransomware and make an analysis of its capabilities in terms of overall and of course early detection. To perform that, the model could be deployed in a sandbox environment, such as Cuckoo Sandbox, and at the same time a ransomware could be executed, such as WannaCry, Petya, Cerber and more. A following comparative analysis on these ransomware's findings could explain the generalizability of the model, its applicability on real-world environments and finally any parts that need to be refined and improved.

Moreover, a training dataset comprised of samples gathered from many different users and devices should be gathered, to accommodate for the different behavioral patterns exhibited by different operators and their devices. Although the monitoring of new subjects and the data gathering process can be simple, there is a possibility that architectural changes may need to take place. As the data gathered are expected to be different than the data of this work, different architectures of the Auto-Encoder may have to be explored or even different models. Ultimately, this extension of the system will be able to encapsulate the legitimate and diverse behavioral patterns of miscellaneous users and devices and minimize the number of false alarms raised when exposed to different environments, focusing solely on the detection of attacks.

As a final point, bringing this work even closer to industrial applications, a new system could be built that considers a wider range of methods that attackers use to conduct their attacks. Different ways to terminate processes, to gain persistence, to exfiltrate data or to encrypt and generally render devices inaccessible to users could be considered and be used to devise a system that tries to capture attacks that employ a broad spectrum of techniques. Also, as this system was created to defend only Windows machines, it is worth extending the system to defend more devices, such as Linux, MacOS, or IoT. Even though these devices are rare targets (although ransomware that target Linux systems are on the rise), the creation of such defensive systems early could add substantial value to the protection and safeguarding of persons, organizations, and institutions.

8 CONCLUSIONS

This dissertation aimed to improve the field of cyber security and specifically the defense infrastructure against ransomware attacks. It strived to do so by creating an anomaly detection system that can detect ransomware attacks in their earliest stages, before the files of the victim are harmed or stolen.

By thoroughly researching current research works and analyzing their shortcomings and their strengths, I proposed to create a system that attempts to fill the gaps of these works and combine their advantages. I first studied and created a diverse dataset from a combination of features that can accurately describe and differentiate ransomware behaviors from normal behaviors. To create that dataset, I monitored a test system for an extensive period, during which the device was used to generate sufficient legitimate and normal traffic, and the features were extracted using Process Monitor and a custom script.

Subsequently I trained an anomaly detection model, specifically an Auto-Encoder, to establish a baseline for the normal behavior, which is used to distinguish normal operations from malicious, and I chose appropriate threshold values, which during validation of the model, exhibited the most promising signs that would perform the best during the testing time.

To test and evaluate the model, a series of experiments were devised and performed, where a custom ransomware attack was developed, following the most important steps taken by ransomware, and deployed on the same test system while the same test device was used and monitored. Once the attack was concluded, the test datasets of these experiments were created and fed into the trained Auto-Encoder to attempt to detect the anomalies exhibited by the attack.

The series of experiments conducted aimed to test the detection capabilities of the ransomware in different settings, such as during times where heavy normal interaction is anticipated and the contrary, and additionally, they aimed to explore the performance of the model when the ransomware deployed an incrementing series of defense evasion techniques, as observed by some ransomware met in the wild.

During these experiments, the ransomware early detection system succeeded in two main areas. Firstly, it managed to generate no False Positives for all the experiments conducted. This outcome means that the detection system can correctly classify normal operations. This is important as False Positive alerts can flood the security analysts with alarms that eventually do not relate to any ongoing attack. So since False Positives can inhibit the efforts of the security teams, the developed system allows the security teams to focus entirely on legitimately malicious behaviors as it generates no such alerts.

Secondly, the system succeeds in the early detection of ransomware attacks for all the experiments performed. Specifically, during the experiments where the ransomware deployed performs all actions in a single process, the detection system managed to detect ransomware attacks very early in the kill chain, when the ransomware attempts to terminate various possibly inhibiting processes, which happens before any encryption or data exfiltration occurs. This behavior validates the hypothesis that a diverse feature set comprised of attributes that relate to the primary stages of the attack can detect ransomware attacks early. In fact, by combining a plethora of features that capture the first steps taken by ransomware, the proposed system captured the attacks once they try to disable the defensive mechanisms of the victim's device or when they attempt to gain persistence by modifying the registry, steps which are done before any encryption or data exfiltration occurs.

However, as the ransomware attacks become more sophisticated and employ more defense evasion techniques, such as avoiding the usage of Windows Encryption DLLs, and spawning other processes to kill the defense mechanisms of the victim's device or to gain persistence, the efficiency of the model tends to drop. In particular, as more defense evasion techniques are used, the ransomware fails to detect the very first steps of the attack, with the most notable case being the attack being captured once the process is added as an AutoRun program via registry key modification.

Despite that drop of efficiency, the detection model yields promising results as it can still capture the attacks in their various pre-encryption stages. Although this work didn't focus on the incorporation of defense evasion techniques but was based on the general activity exhibited by single-process ransomware, this series of experiments showed that this feature set is capable of successfully capturing attacks before encryption of the files has started, even when attacks that utilize multiple processes are conducted.

Another trend that was observed is that in general the developed detection system was more efficient during the non-working hours of day, where less activities were recorded by normal processes in comparison to working hours. This behavior is aligned with the expected one, as ransomware attacks are labor intensive and can be distinguished easier when the victim's device is not operated heavily.

In conclusion, the created anomaly detection system, by leveraging an Auto-Encoder, succeeded in the early detection of ransomware attacks, the main objective of the research. It captures a variety of ransomware attacks during their pre-encryption stages and simultaneously produces no False Positive alerts, allowing the security teams to focus exclusively on the alerts that are associated with ongoing ransomware attacks.

9 REFLECTION

I can confidently say that this work has been immensely beneficial for me, especially as over the past years I have been focusing and specializing in Cyber Security and applications of Machine Learning in it. From the time that I started working on this project, it was apparent that it is a topic that greatly troubles both the industry and the research community as ransomware tend to cause serious harm to its victims, especially since its exponential rise during the recent years. Therefore, I had the opportunity not only to study the nature of a current and worrying problem, but also to provide a cutting-edge solution that can advance the research and creation of novel defensive mechanisms.

Initially, by researching the nature of ransomware I managed to get a deep look into the character and motives of ransomware attacks. I achieved a great understanding of the various tactics and techniques that attackers employ to create robust ransomware and maximize their impact, and additionally how they may attempt to evade the defenses of current implementations. Also, as a part of the effort to understand the operation of ransomware, I meticulously observed the underlying effects of attacks on the infected systems, not only in terms of the damage inflicted but on less visible traces that expose an attack and finally enable its detection in its early stages.

The study of the previous research and implementations allowed me to fully grasp the current approaches to ransomware detection. With that study, I was able to constructively address the limitations of these techniques, and to highlight their promise, which prompted me to devise ways to create my own implementation, remediating some of the identified drawbacks and combining their strengths.

Furthermore, I explored a variety of state-of-the-art machine learning algorithms, and I got a deep understanding of anomaly detection methods. Most importantly, I studied Auto-Encoders, their architecture, their use cases, their strengths, and drawbacks. Also, I explored and experimented with a variety of implementation choices and applied best practices followed in machine learning projects. This was extremely advantageous for me as I comprehended and succeeded in implementing the demanding workflow of a vigorous machine learning project.

Moreover, throughout this project I managed to provide high-quality results in a short amount of time, being diligent to various deadlines and to be flexible in any changes that needed to be made along the way. My communication with my supervisors was instrumental, as under their guidance, at many points I had to think critically about my approaches and my work, and finally devise ways to amend any issues that I was facing. Since this dissertation tackled a problem that significantly distresses professionals and researchers, it follows that its exploration and solution will have equally hard difficulties to overcome. However, throughout our meetings and with thorough independent research, I managed to get a grasp of possible solutions, test them, and finally successfully implement them.

Last but not least, the nature of this dissertation enabled me to develop a more methodical and organized way of working, based on planning, consistency, resilience and plenty of constructive self-criticism. All these greatly contributed to the overall result of this dissertation and will be undeniably valuable for any subsequent projects and research.

In summation, the study and the creation of this system was genuinely advantageous, inspiring, and fascinating in many different ways and I am confident that I am capable of successfully researching in-depth state-of-the-art topics, combining knowledge from different fields and implementing solutions to bridge gaps identified in the research and industry community using cutting-edge technologies.