**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 a very low amount of false positive alerts.

Acknowledgments

Contents

[1 Introduction 7](#_Toc141919704)

[1.1 Motivation 8](#_Toc141919705)

[1.2 Contributions 9](#_Toc141919706)

[1.3 Structure of the Dissertation 10](#_Toc141919707)

[2 Background 11](#_Toc141919708)

[2.1 Ransomware 11](#_Toc141919709)

[2.1.1 Targets of Ransomware 13](#_Toc141919710)

[2.1.2 The Nature of Ransomware; The Ransomware Kill Chain 14](#_Toc141919711)

[2.2 Anomaly Detection 18](#_Toc141919712)

[2.2.1 Anomaly Detection as Learning Normal Behavior 19](#_Toc141919713)

[2.2.2 Taxonomy of Anomaly Detection Methods 20](#_Toc141919714)

[2.2.3 Evaluation Methods 21](#_Toc141919715)

[2.2.4 Auto-Encoders 23](#_Toc141919716)

[2.2.5 Architecture 24](#_Toc141919717)

[2.2.6 Training an Auto-Encoder 24](#_Toc141919718)

[2.2.7 Anomaly Scoring 25](#_Toc141919719)

[2.3 Detecting Ransomware 25](#_Toc141919720)

[2.3.1 Static Analysis 26](#_Toc141919721)

[2.3.2 Dynamic Analysis 26](#_Toc141919722)

[2.3.3 Deception-Based Detection 26](#_Toc141919723)

[3 Literature Review 27](#_Toc141919724)

[4 Methodology 31](#_Toc141919725)

[4.1 Dataset Creation. Maybe combine with the dataset collection section 31](#_Toc141919726)

[4.2 Detection Model 37](#_Toc141919727)

[4.3 Establishing a Normal Baseline 38](#_Toc141919728)

[5 Experiments 42](#_Toc141919729)

[5.1 Experimental Setup 42](#_Toc141919730)

[5.2 Dataset Description 42](#_Toc141919731)

[5.3 Preprocessing and Data Preparation 43](#_Toc141919732)

[5.4 Hyperparameter Tuning 44](#_Toc141919733)

[5.5 Threshold Choices Exploration 47](#_Toc141919734)

[5.6 Training 49](#_Toc141919735)

[6 Results 50](#_Toc141919736)

[7 Discussion and Limitations 51](#_Toc141919737)

[8 Conclusions 52](#_Toc141919738)

[9 Reflection 53](#_Toc141919739)

List of Figures

[Figure 4‑1: The Ransom Note dropped in the directories of the victim. 35](#_Toc141815739)

[Figure 5‑1: The Auto-Encoder's final architecture 39](#_Toc141815740)

# 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.

## 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](https://doi.org/10.3390%2Fs21134294) 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.

## 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.

## Structure of the Dissertation

# 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 Anomaly Detection Systems, Auto-Encoders and finally the various methods that are employed specifically for ransomware detection. After reading this chapter, the reader will be able to grasp the what are ransomware attacks, Auto-Encoders and what are they used for, and usual techniques employed to detect ransomware.

## 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 and may additionally threaten to publish the victim’s data on the Internet unless a ransom is paid to the attackers, in what is called a “double extortion” attack. 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 exfiltrate files that the attackers may find useful, to servers that they control and threaten the victims to publish them, unless that ransom is paid. Since the data that are encrypted usually relate to core business processes, contain vital information and the assets that are compromised are crucial for the normal activities of the victims, their inaccessibility puts pressure on the victims 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’ data. 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.

Another notable mention is *Scareware.* Scareware, as the name suggests, works by scaring the victim into paying the attackers an amount of money. Victims may be bombarded by messages and pop-ups saying that their systems have been infected by a ransomware, that their files have been encrypted or that they have been lost access to their systems, and that the only way to regain access is to pay an amount of money. In reality though, this type of malware does not actually affect the machine and counts only on the fear and panic of the user to pay the ransom.

Ransomware have generally been around since 1989, when the first incident classified as a ransomware, the PC Cyborg attack, was documented. With the passage of time and 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 and targeting predominantly Windows machines. Additionally, early ransomware were using exclusively symmetric ciphers to encrypt the target files. GPCode was infecting machines via phishing emails and utilized custom symmetric encryption algorithms and storing the encryption key at the victim’s machine. 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 demanded payment via a paid SMS.

Nowadays, ransomware utilize more sophisticated encryption and extortion methods, employing hybrid encryption and double exfiltration. 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 and the confidentiality offered by asymmetric ciphers. In a double exfiltration attack, attackers do not only encrypt the victim’s files, but they also threaten to publish the files online, unless the 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 and 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 not 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 the largest and most vital pipelines in the United States. Instead of threat actors developing their own attacks, they leveraged the RaaS offered by DarkSide and performed one of the most largest publicly disclosed cyber-attacks against critical infrastructure in the US.

### Targets of Ransomware

When it comes to ransomware there isn’t a specific group that can feel not threatened. 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 of it often takes 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 workstation 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 be affected ransomware, but 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 sought after devices.

Although the targets can be diverse, the most affected type and operating system are PCs/workstations running Windows. As a result of the high popularity of Windows among users, attackers favor Windows 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 Windows operating system as most ransomware target Windows. Therefore, for the rest of this work any mentions to ransomware are supposed to be crypto-ransomware that target Windows.

Talk about:

* Evolution of Ransomware-Polymorphic

### 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 must conduct reconnaissance activities to gather intelligence about the victims, as ransomware are usually targeted and do not rely purely on the random spread of the malware. 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 IPs 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 in a drive-by compromise. For example, Ryuk uses phishing campaigns which redirect unaware users to documents hosted in legitimate hosting services, such as Google Drive. Additionally, they may be dropped by other malware, if the victim has been infected by another malware cooperating with the ransomware. Furthermore, the delivery can be conducted by Supply Chain attacks, where a trusted third party is compromised and used to distribute malware.

**Exploitation and Privilege Escalation**

Many of the actions that ransomware need to take (deletion of shadow copies, halting of certain processes and utilities, interaction with the Windows registry, access to specific directories and files) require elevated privileges. To get escalate their privileges, the malware may exploit vulnerabilities, such as zero-day 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 the Windows UAC prompt. While vulnerability exploitation is the most preferred way to gain elevated privileges, ransomware such as Sodin, uses UAC prompts to request elevated privileges, if exploitation is not possible.

**Installation**

Ensuring that the malware has sufficient privileges to perform its actions, it unpacks itself and decrypts its payload so that its files and are accessible to be executed and used for the attack. Then, to ensure that only one instance of the malware is active on the victim’s machine it may use a mutex and check. Next, the ransomware check if the machine is included in a whitelist, that is machines that should not be attacked, as explained in the Reconnaissance phase. If the machine is included in the whitelist, then the attack is aborted.

Subsequently, ransomware tend to terminate processes that may detect malicious activity, inhibit file encryption and file access and 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, the ransomware tries to gain *persistence*. To do that, it is common that ransomware 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 a Command and Control server 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(s), which may be done via custom protocols, encrypted channels, 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 send 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 discovered target files and some access also 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, many file types are not targeted by ransomware, such as critical Windows files, web browsers and executables, as they are used by the user to minimally interact with the device, to get notified about the infection, to pay the ransom and 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(s). 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”.*

Having encrypted all the desired files, 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 Windows Recovery Environment, so that recovery of the system is impossible.

**Extortion**

To conclude the attack, the attacker notifies the user that his files are encrypted by placing a ransom note in each encrypted 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 of the ransomware that are met in the wild 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.

## 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 given 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 number 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 of 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 does 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.

### 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 the 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 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.

### 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 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 the anomalous data are very rare.

### Evaluation Methods

As explained in the previous section when building anomaly detection systems, it is expected to deal with datasets suffer from class imbalance. The most intuitive way to evaluate the performance of a model is to calculate the percentage of its correct predictions. That percentage of correct predictions is called accuracy of the model. But in such models, simple metrics like the simple 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 is 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 that 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 anomalies, which correspond to the notion of existence and therefore positive labels. 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 help shift the focus to 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 and harm the target device.

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.

Inversely, The FNR can also be calculated as 1-TPR and the FPR as 1-TNR. The formulas of these metrics are presented below:

Need a smooth transition from anomaly detection to autoencoders

### 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. Since the output is the same as the input and they don’t need any labels to train, Auto-Encoders belong to the category of unsupervised learning algorithms.

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 used to infer that that particular input corresponds to an anomaly.

### 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 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 size of the bottleneck, that is the number of neurons 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.

### 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 of which the model observed 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 a sufficient number of 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.

### 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.

## Detecting Ransomware

### Static Analysis

### Dynamic Analysis

### Deception-Based Detection

# 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.

Finally, <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.

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 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.

# 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 minimizes the number of false positive alerts generated.

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 such 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. Instead of using an already existing dataset, 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.

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 mathematically uncover the patterns and trends of a normal behavior.

To test the detection capabilities of this model, I advise to develop a ransomware that performs the main actions of ransomware following the presented 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.

## Dataset Creation. Maybe combine with the dataset collection section

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 help 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.

Before diving into the exact features that will be used, it is important to reiterate that no feature can detect ransomware attacks on each own, but different features are combined to create a comprehensive *set* of features capable of describing and ultimately detecting a ransomware infection.

Initially, many actions performed by ransomware require elevated privileges and for that reason during the first steps of the execution, ransomware processes try to gain elevated privileges. While it is hard to enumerate and capture all methods that a process might gain such privileges, 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, eventually this privilege escalation will be noticeable by the change of the process’s status.

Progressing in the ransomware kill chain, ransomware try to gain persistence by modifying the Windows Registry AutoRun keys. Specifically, ransomware may use 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.

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. In such scenarios, the process that modifies the registry and the main body of the malware have different Process IDs and their cooperation cannot be easily traced. To accommodate that, I advocate to include another feature which states whether a process exists already in the Registry AutoRun keys.

Subsequently, ransomware try to terminate certain processes that inhibit file access, encryption or that relate to defense systems such as antivirus software, firewalls etc. As this is a key step in a successful attack, I suggest monitoring the use of certain applications and tools that kill processes, such as “TASKKILL”. Normal applications use such tools and commands very rarely and such events should be included as a part of the feature set that can differentiate normal from ransomware activity.

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.

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, regardless of the type of C&C or exfiltration server employed.

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 use of encryption by inspecting the side effects of certain activities. For instance, ransomware are anticipated to write to many files in a short amount of time, especially since attacks as usually sped 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 mainly specific 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 and other indirectly linked stakeholders, and any other data and information that may disrupt the normal operation of the victim’s activities. 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.

Additionally, in information theory, “Shannon Entropy” is a measure of uncertainty or 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, where 0 indicates no randomness and 8 indicates complete randomness in the given sequence. Therefore, as encryption generates seemingly random sequences of data, it in consequence produces high entropy data and thus high entropy can be used as a feature to infer encryption. Because ransomware are expected to write high entropy data to a large number of files during a given time period, it is then advisable to calculate the average entropy of the files that a process wrote to, additionally to the number of files that were modified by that process.

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 overwrite the contents of the files with the encrypted data and never write any data to them again. Supplementarily to the previous two features, this can help distinguish normal from ransomware activity.

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 process 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 behaviors similar to a ransomware for some features, the *combination* of all the features is what will definitely differentiate normal from suspicious behavior. For example, a ransomware is expected to send some amount of data over the web and have elevated privileges, and have accessed AutoRun Registry keys in the past *and* has written to many files with high entropy etc. Therefore, the *combination* of those otherwise normal feature values, will eventually classify a behavior as ransomware and not normal.

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.

**DATA COLLECTION**

To gather the necessary data, the test system must be monitored for a specific period and across different time contexts. In this work, I used the Process Monitor utility (or procmon), an advanced tool for Windows that can monitors and records changes made in the system by all 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.

Firstly, 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 being a working hour and any time outside of that I record it to be a non-working hour.

I used procmon and extracted the volume of the egress network traffic from the “Detail” column, information useful to detect any data exfiltration and C&C communication attempts.

Then, I monitored *any* interactions of the processes with the AutoRun registry keys, to detect any attempts of processes to gain persistence. It is prudent to monitor any interaction with these keys as different ransomware may initially simply check if they exist already in the registry by relying on another process to make that modification.

To supplement that, I monitor if a process is already registered as an AutoRun program, by another process, or by any other means. To do that, I utilize procmon’s “Command Line” column which lists the command that was issued to execute that process, strip any command line arguments that may have been for execution and check if the remaining executable paths exist in the registry AutoRun keys.

Additionally, when a process traverses directories and enumerates the files in that directory, it generates “QueryDirectory” calls. Therefore, I record such calls made by each process by using the “Operation” column. As mentioned before, a large volume of such calls can be indicative of a ransomware trying to discover files to encrypt by traversing the directories of a system.

Next, I track the number of files written by a process in a certain amount of time, by recording the “WriteFile” operations and using the “Path” column to infer the file that the process wrote in. Using those two pieces of information I can get the number of files that a process wrote to for example, when a ransomware writes the encrypted data in a file. It must be noted, that I do not blindly record write operations in any file type, but I focus on file types that are usually targeted by ransomware, such as documents, media, compressed files and others, as explained earlier. A large number of write operations in a short amount of time can signify a ransomware infection.

An important feature is the average written entropy. As described before, a large entropy of a file can be an indication that a file is encrypted or compressed, due to the nature of these methods. To capture the average *per-file* written entropy by a process I calculate the entropy of the files that were identified in the previous feature, and I use the following formula to calculate their data’s entropy. Finally, I average the calculated entropies of all the files, to get the average entropy of all the files that were written by each process. MENTION CHOICE OF 4 DEFAULT ENTROPY. EITHER HERE OR IN ANOTHER SECTION

Usually, the use of encryption also means the usage of DLLs related to cryptographic functions. Using the process activity utilities of procmon I keep track of efforts of processes to load cryptographic DLLs, by monitoring any “Load Image” operations. In detail, I take into account only DLLs that contain the word “crypt” inside, such as the “bcrypt.dll”, the “crypt32.dll” and other DLLs that are part of the various Windows cryptography APIs. Undoubtedly, attackers may bypass the use of such APIs by statically linking encryption DLLs. As Stamper explains though Stamper, API and DLL monitoring should be used as a *part* of a larger feature set.

While procmon is a tool that provides access to granular information, it has its limitations. An important feature to keep track of, is whether a process is running with elevated privileges. To get hold of that information, I created the following *Powershell* script influenced by [CITE SCRIPT INFLUENCE SOURCE HERE], that iteratively enumerates all the processes running in a system and their privileges.

At the end, by running the script simultaneously with procmon I get all the active processes in the system and their privileges, gathering all the necessary information to create the dataset.

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 |

Table 2: Features used in the dataset.

## Detection Model

For an accurate and reliant detection system, equally important to the features used is the model that is deployed. In this work I propose to use an Auto-Encoder to learn a representation of the operation of normal applications. After the normal behavior has been baselined, the same model will be used to infer if the behavior of a process is an anomaly or not.

Although more sophisticated models and variations of the Auto-Encoder 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 encoder learns to recreate the input data from that compressed representation.

In this work, the Auto-Encoder will learn the patterns and trends of normally behaving processes, by learning to compress the input features into a latent representation and it will learn how to effectively reconstruct the original features from this latent vector. By minimizing the reconstruction loss 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.

For the Auto-Encoder to learn accurately the intricacies of a normal operation and to work appropriately it is important that the training dataset and validation datasets include exclusively samples from normal applications. Thus, it is crucial to ensure that the device that is used to collect and create the dataset is not infected with any kind of malware and specifically with ransomware, as the inclusion of ransomware samples in the training dataset will force the model to learn how to compress and reconstruct effectively malicious events as well.

That being said, 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 not been trained on any ransomware infected instances. Generally, the output of the Auto-Encoder will be the reconstruction loss of the input given. It is expected that normal inputs, that is samples that correspond to normal behavior, will have a low reconstruction loss and abnormal inputs, will result in a generally higher reconstruction loss. Using that reconstruction loss 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 loss 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 and belongs to the hyperparameter tuning phase of the system, it must be said that there are many choices for thresholds that can influence the efficiency of the model.

## 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 that will be collected must reflect a real use of the system where plenty and realistic interaction of the user with the system takes place.

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 use the device normally, performing 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 activities that they normally do in the system.

Additionally, the device should be monitored across different times of the day, to capture the behavioral patterns in both working and non-working hours of the day. This will increase the expressive capabilities of the model and it will enable it to be used across different times of the day.

It is imperative that while the system is monitored to baseline normal, the device is not infected with any kind of malware and most importantly with ransomware. As the Auto-Encoder that will be trained is used to capture and model only the normal behavior, the training data should not contain any malicious processes that can reduce the perception of normal and anomaly detection capabilities of the system.

**RANSOMWARE DEVELOPMENT**

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, which were described in a previous section. 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 subsystem. Of course, the ransomware could be written in any other programming language, but Python offers great capabilities for the development 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. Therefore, I assume that the attackers have managed to deliver the payload to the target machine and have managed to unpack and decrypt its contents. 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, this ransomware tries to conduct the attack on its own, 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 checks if it has elevated privileges (exploitation phase). If it does not, then it tries to elevate its privileges with a UAC prompt by directly asking the user to grant those privileges. If the user grants them, then the process is restarted with elevated privileges. This choice was made as the feature that I chose to use 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 run in elevated mode. This way, it can capture any technique used by an attacker to elevate the privilege of a process.

Having acquired elevated privileges the script tries to gain *persistence* (installation phase) 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.

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 all files indiscriminately. On the contrary, it is configured to encrypt only a specific set of file types, by examining their extensions. Specifically, the file extensions targeted are listed in the table below. Overall, the ransomware aims 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 and other indirectly linked stakeholders, and any other data and information 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. 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, as was explained earlier, AES was used as the preferred cipher as it is a strong encryption algorithm, and additionally it is symmetric, which means that it will be able to encrypt a big amount of files in a short period of time, in comparison to asymmetric ciphers, directly corresponding to robust ransomware observed in the wild.

When the key is generated, it is sent to the C&C server so the C&C can provide the decryption key once the ransom is paid, and the victim can decrypt their assets. WRITE ABOUT ENCRYPTING WITH ASYMMETRIC KEY! When the key is transmitted, the ransomware connects with another C&C server that is in charge of data exfiltration. This server instead of communicating over a custom protocol (as the previous one) is a configured HTTP(S) server. Data exfiltration to an HTTP/S server is a very common and crucial method when performing data exfiltration attacks, because it can give a significant amount of cover due to the likelihood that users of the machine are already communicating with such protocols over their normal operation of their device. 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.

Once all the files have been exfiltrated to the HTTP 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 (as not all data stored in a file is expected to be in text 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 ways, such as searching for directories containing the word “backup”, deleting shadow copies, but this method used here suffices to simulate the complicate and make the recovery of the system harder.

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.



Figure 4‑1: The Ransom Note dropped in the directories of the victim.

# Experiments

Introduction:

* What I did:
  + Data preparation and preprocessing

## Experimental Setup

* Description of the system
  + OS
* Monitored the system

## Dataset Description

The final dataset that was used for the creation of the system is described in this section. As was stated before, it comprised of the features outlined in the table 1. It consists of X samples and eleven features that correspond to the various stages of ransomware attacks. This feature set consists of both numerical and categorical/binary features.

* Number of samples
  + Working
  + Non Working
* Number of unique samples. State that it is important to keep only one instance of the samples
* Statistical description of the dataset. Maybe present them as a table and just explain some stuff, that some are in different ranges (net, entropy and stuff)
  + Numerical
    - Max/min
    - Mean
    - Std
  + Binary
    - Counts for each class

## 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 not numerical but categorical and specifically binary (0 and 1). For that reason, it is considered best practice to use One-Hot encoding for those features. So, every binary feature in the dataset is transformed into two features in the following way:

A screen shot of a computer

Description automatically generated

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

It is also important that the dataset does not include duplicate values, as the model may overfit and be heavily influenced to learn mostly these duplicate values. For that reason, I discarded all the duplicate samples and kept the unique samples only.

Next, 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 80/20.

It can be seen that the numerical features used have different scales. For example, the “NET” feature ranges from 0 to tens of thousands, while “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 sample was 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 will be similar.

## 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:

|  |  |
| --- | --- |
| Equation 1:Mean Square Error (MSE) function | Equation 2: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 usage 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 on only normal behavioral data that may contain some rare and “abnormal” legitimate data points, these “outliers” should be treated as such. Therefore, the 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 chosen as the loss function of the model as it performed much better, providing more granular results and interpretations of the differences between 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:

A screenshot of a computer

Description automatically generated

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

The Encoder (and the Decoder in reverse order) overall has X layers, one input layer, X hidden layers and one bottleneck layer, the layer of the target latent representation. Its input layer has X inputs neurons, one for each of the input features used. The first hidden layer comprises of X/2 neurons, which connects to the next hidden layer, which in turn has X/4. The final hidden layer consists also of X/4 nodes, keeping the number of neurons the same with 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 X neurons and therefore X output values which are fed to the loss function chosen to calculate the reconstruction loss of the model.

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.

In connection with the batch size, I explored the number of epochs that the model will be trained for. An epoch is a single pass through the entire dataset and the number of epochs define the number of times that the model will see the dataset. It is important that the number of epochs is set correctly, as it can affect the generalization of the model. 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 training several times, 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 0.3 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 overfitting signs and therefore it was deemed unnecessary.

## 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 therefore an alarm should be raised 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 generated, 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 + (or -, although in this case we disregard -) “x” time sigma (σ), where σ is the standard deviation 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 outliers which should be classified as abnormalities 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, I favored sigma rule-based thresholds for this work.

Originally, sigma rules are used for data that are normally distributed and one might think that they are not suitable for this case. Nevertheless, I advocate that sigma rule-based thresholds are suitable for this work. 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 and very few samples will result in high loss, primarily those belonging to ransomware behaviors. Therefore, normal data may not exist above a defined standard deviation, leading to minimal or even no false negatives generated. For those reasons, a proper choice of a standard deviation from which the sigma rule will be calculated for the threshold 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 5-sigma rule which produced no false negatives. Of course, larger standard deviations also produce no false negatives. 5 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.

## Training

# Results

* Not hyperparameter tuning. Everything about architecture, optims, losses etc shiuld be stated in experiments

# Discussion and Limitations

# Conclusions

# Reflection