POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Analysis and improvement of ransomware detection techniques

Supervisor:

Prof. Cataldo Basile

Company Supervisor:

Riccardo Cardiello

Candidate:

Marco Smorti

Academic Year 2022/2023
Torino

Abstract

Ransomware is a malware that is able not only to disrupt normal behavior of a system, but the most sophisticated ransomware can attack entire networks, leading to service interruption mainly due to data encryption that makes data useless, which means a loss of money for a company most of the time. Having backups is extremely important, but we want to be able not only to recover from the issue, but also to detect the attack as soon as possible and to stop it. Since ransomware leave traces in the network, we want to exploit this in order to perform the actual detection. The focus of the thesis was first to perform an introduction to ransomware and an analysis on most popular ransomware in Italy in 2022. After understanding what was currently detected by the existing network intrusion detection system that has to be improved, the next step was then to perform an analysis on existing network traffic captures that recorded a ransomware attack being performed. This was done in order to understand the reports generated by Zeek, that is a network traffic analyzer that gets as input raw traffic capture data and generates many types of reports, but those existing captures were not sufficient to perform the actual detection. Since real ransomware are too dangerous to intentionally run them in a testing environment, an ad-hoc ransomware has been created using Python adopting a different encryption technique with respect to the techniques of the most popular ransomware that are already detected by the system. Since this kind of ransomware was not detected by the existing network intrusion detection system, it has been used by running a simulation of a network attack using virtual machines and SMB protocol and by recording the traffic using Wireshark, in order to later analyze the generated traffic. At this point, the analysis and detection were performed using the Spring framework based on Java. The detection starts from the Zeek reports generated from raw traffic captures and it is

based on the frequency of SMB operations over time and on percentiles that are computed on past traffic and used as thresholds for detection in newly analyzed traffic. Since this was based on a simulation, the generated code was then adapted to the real implementation carried out by the network intrusion detection system, using the real traffic received as input.

Ringraziamenti

Ci tengo a ringraziare in primis *aizoOn* e specificatamente *Simone Janin*, *Riccardo Cardiello* e *Federica Bisio* per l'opportunità che mi hanno dato di svolgere questa tesi ed il supporto fornitomi durante tutto questo percorso. Ringrazio inoltre il mio relatore, *Cataldo Basile*, per aver accettato la mia proposta di tesi e per avermi supportato nello svolgimento della stessa.

A mia madre Marisa, mio padre Antonio e mia sorella Lucia, fonti di ispirazione e colonne portanti della mia vita. Grazie per aver sempre creduto in me e per avermi aiutato a inseguire i miei sogni.

Ai miei zii e parenti tutti, vicini e lontani, grazie. Grazie, perché anche il vostro sostegno costante e sincero mi ha portato fin qui e mi ha fatto sentire sempre come a casa.

A Tommy, incontro del destino trasformatosi nel compagno perfetto di questo viaggio che in sostanza abbiamo fatto insieme e in cui non è stato facile sopportarmi nelle mie paranoie universitarie. Grazie.

Ad Antonino e Daiana, un bellissimo incontro del mio viaggio voluto dal destino, trasformatosi poi nella mia seconda famiglia. Grazie per avermi sempre motivato e non avermi permesso di mollare mai.

A Gabriele e Mattia, ogni giorno sempre più contento di avervi incontrato. Soltanto noi sappiamo quanto sia stato difficile e tortuoso questo percorso, ma è solo grazie alla squadra che siamo diventati se oggi posso scrivervi queste parole. Grazie.

Ad Erika e Federica, compagne di vita che sento presente sempre e da

sempre. Sono 15 anni che continuate a darmi forza e a sostenermi in ogni mio momento, bello o brutto che sia. Grazie, sorelline.

Ad Alessio, Andrea, Chiara, Samuele e Vincenzo, e chiunque io abbia incontrato in questo viaggio. Grazie, perché da voi ho sempre ricevuto ottimi consigli ed ognuno di voi ha contribuito a rendermi ciò che sono.

A "quelli di Ing. Inf", una community fatta da persone splendide che porterò per sempre nel mio cuore. Grazie.

Table of Contents

List of Tables

List of Figures

Acronyms

NIST
National Institute of Standards and Technology

SMB
Server Message Block

RaaS
Ransomware-as-a-service

JSON
JavaScript Object Notation

MD5
Cryptographic hash function

pcap
Packet Capture

TCP
Transmission Control Protocol

UDP
User Datagram Protocol

DHCP
Dynamic Host Configuration Protocol

IP

Internet Protocol address

MAC

Media Access Control address

DNS

Domain Name System

MIME

Multipurpose Internet Mail Extensions

FUID

File Unique IDentifier

GUI

Graphical User Interface

URI

Uniform Resource Identifier

CSV

Comma-separated values

C&C

Command & Control server

OS

Operating System

VM

Virtual Machine

RAM

Random Access Memory

Chapter 1

Introduction

One of the most disruptive malware that is spreading nowadays through the Internet is ransomware. Thanks to the carelessness of users, ransomware can mainly spread thanks to some phishing techniques such as spam emails containing the malware as attachments. When a user opens it, there can be huge damage such as data loss, which means that user data is encrypted and no longer usable until a ransom asked by the attacker to the victim is paid. Paying a ransom is illegal, and even if we pay it, we're not sure that the attacker will give us the key to decrypt all user data. It is clear that the main targets of attackers are companies and corporations since in this case data has a high value and the encryption of that data can also cause service disruption and money loss for those companies. A possible countermeasure to this kind of attack is backup, but backups struggle with some limitations such as the validity of data strictly related to the frequency in which backups are performed. For this reason, we want to be able not only to recover from a ransomware attack, but we want to detect that it is performing encryption of data to stop it as soon as possible.

Thanks to the collaboration with *aizoOn Consulting S.r.l.* it has been possible to access an existing network intrusion detection system, capable of detecting ransomware attacks through the network. The scenario is one of the enterprise networks, in which files are usually stored on a server and shared between users using a protocol. One existing approach for detection is based on changes in some bytes at the beginning of files, which is usually performed

by some ransomware, which alter the so-called MIME type. Since not all ransomware work in this way, but others leave the MIME type untouched by encrypting only the content of files, the previous approach is not always valid. The first part of the thesis focuses on the analysis and understanding of what kind of ransomware the system detects and how the detection works. The second part is then focused on improving the detection and creating an ad-hoc ransomware script.

After a brief introduction to ransomware and some statistics about their diffusion, the used tools are introduced. Then, there is the analysis of the results of ransomware traffic captures found online and given as input to software that can perform traffic analysis. Then, a specific kind of ransomware that is not recognized by the system has been created using Python. A testing environment using Ubuntu virtual machines with an SMB server and client installed has been set up to generate a traffic capture containing the developed ransomware in action. This was done to run the ransomware in a secure and controller way and to sniff the network traffic. Later on, this capture was then analyzed to create the Java code that can perform the detection based on what is possible to understand from the sniffed traffic. In the end, the conclusion on the possible evolution of the code is reported.

1.1 Structure of the document

The document is organized as follows:

- Chapter 2 - Brief introduction to cybersecurity and ransomware: description of concepts that will be the basis for understanding the thesis. This chapter contains also an analysis of ransomware diffusion in Italy in 2022.
- Chapter 3 - All the material needed for the work is presented here. Here are introduced and briefly described the tools for analysis and detection.
- Chapter 4 - The state of the art: here there is the analysis of what is actually implemented by the existing system and what resources have been found online.

- Chapter 5 - Ransomware design and development: since online resources are not sufficient, but also for security reasons, in this chapter, there is the design, development, and analysis of a custom ransomware script that meets the requirements.
- Chapter 6 - Detection: this chapter implements the detection of the previously developed ransomware using Java.
- Chapter 7 - Conclusion: in the end, this last chapter summarizes the achieved results and draws the conclusions of the thesis.

Chapter 2

First look to cybersecurity and ransomware

The advent of modern technologies that are capable of extremely complex computations has completely changed our way of life and it is constantly evolving allowing us to simplify many processes in any field. Companies all over the world have many kinds of devices that are connected to the internet and use them to provide services. The possibility to interact between different devices worldwide must let us think not only about the good behavior that users can have but also about the malicious ones. Companies must define what assets are, that is, what has value and what needs to be protected, and then must define adequate protection measures for those assets.

2.1 What is cybersecurity?

According to NIST, cybersecurity is the process in which there is the prevention of damage and the protection of any device and communication, including information contained therein, that may come from any external threat.[1] This means that it is possible to define it as cyberattack any attempt from an external malicious user to disrupt the normal behavior of any device or communication.

2.2 Ransomware: definition and overview

Among all the various kinds of cyberattacks, this thesis takes into consideration a specific kind of malware, which is a general term to refer to any malicious piece of code, called ransomware. Ransomware is a malware type that has the specific aim to ask for a ransom from the victim after denying access to some resources (e.g., important files and documents) or even to the whole system, making it unusable. The number of attacks using ransomware is increasing over the years due to many factors such as the fact that the higher the value of information stolen from the victim, the higher the financial return that an attacker can potentially receive. Ransomware can infect many kinds of systems by exploiting vulnerabilities in operating systems and programs, but mostly thanks to the carelessness of users.[2] It is possible to divide ransomware into two main classes:

- **Locker ransomware**: locks the whole target system making it quite unusable. Only limited operations are possible (e.g., it is only possible to pay the ransom);
- **Crypto ransomware**: prevents the victim from accessing valuable data stored on the system. Data is made useless by typically using encryption, so the victim will need the corresponding key to restore its original state. [3]

The second class is much more effective than the first one. The first one denies access to a computing resource, but it is possible to use some techniques to restore the system's previous state without losing data. The second one lets the system work normally since it only affects the data stored on it using encryption techniques. This makes data useless unless the victim has access to the decrypting key that in some cases is given to the victim after the ransom is paid, in other cases money is just stolen and data is lost forever. Our attention will be on the second class. The crypto-ransomware scenario is really wide and among all the ransomware for which it is possible to get a look at the source code, it is possible to notice that all of them have the same aim: encrypt the highest data as possible. Each of them can then use different encryption techniques, encrypt different types of files, force some processes to stop, and so on.

Figure 2.1: Ransomware attack general steps, [4]

In Figure 2.1 is reported the general behavior of ransomware that can be divided into five steps:

- **Infection**: it is the attack vector that allows the ransomware to enter the system. The most common method is sending e-mail-attached files that are executed by the user. It is also possible to directly download the binary from infected web pages. Finally, some ransomware strains implement a worm-like behavior to propagate through a Local Area Network.
- **Contact C&C servers**: some ransomware contact a C&C server to obtain or store the encryption key. This step can also take place after the data encryption if the ransomware works offline, and the C&C server is contacted in the last step to store the decryption key.
- **Encryption key management**: the key can be obtained from C&C servers or it can be created locally, but in this case, ransomware will encrypt the generated key with one obtained from the C&C server and then will store it on the server.
- **Data encryption**: the main task of this kind of malware. The ransomware encrypts and deletes user files. It usually also affects files and volumes mounted using a network file-sharing protocol such as SMB.
- **Extortion**: this is the last step in which the malware requests the payment of ransom to decrypt files. It usually explains how to pay the ransom and sets a deadline and threatens to delete some of the user's files every hour.

The aim is to detect and block the ransomware before the fifth step, since in this case all user data will be already encrypted. The fourth step is the last chance to detect the attack before losing data, and it is where the thesis

will focus, since the tools that focus on this phase usually analyse some parameters obtained from user files such as variations in the content of files, modifications in some bytes of each file, read and write frequency.[4] To better understand why it is extremely important to increase security against ransomware attacks, a report about incidents in Italy in 2022 will show some statistics on the number of attacks.

2.3 Spreading of ransomware attacks in Italy in 2022

As reported by SOCRadar, Italy in 2022 is in the top 10 countries with the most number of posts submitted by ransomware gangs. It is estimated that about 60% of organizations dealt with ransomwares in the first 6 months of 2022. Even if ransom payments are illigal in Italy, it is estimated a total amount of $710.000 ransom payments.[5] The main target of cybercriminals is the manufacturing sector, which demonstrates their capability to deploy the RaaS model[6], that isa business model that let criminals perform ransomware attacks by having limited technical capabilities, since the RaaS platform will supply the malware source code and everything that is needed to adapt and inject it in the target system. As SOCRadar reports, the top ransomware groups of 2022 in Italy are:

- **LockBit 3.0**: RaaS operator. This malware is able to auto-propagate itself thanks to some automated processes that are able to discover new accessible victims, connect and infect them. The first version of this ransomware was born in late 2019, while the last one came out in June 2022. It contains one of the best locker algorithms in terms of speed and overall functionality. The majority of reported incidents in Italy is about this malware.
- **Conti**: RaaS operator. Probably born in Russia in 2020, it started spreading worldwide after the conflict between Russia and Ukraine. While it encrypts all possible files, it tries to delete Windows' Volume Shadows Copies and tries to stop some services in order to be able to encrypt much more files. Some of the source code has been leaked by a Ukranian hacker. This gang is one of the most active nowadays.

- **AlphVM Blackcat**: RaaS operator. It first showed up in late 2021. Unlike the most popular ransomware, this is one of the first written using the Rust programming language.

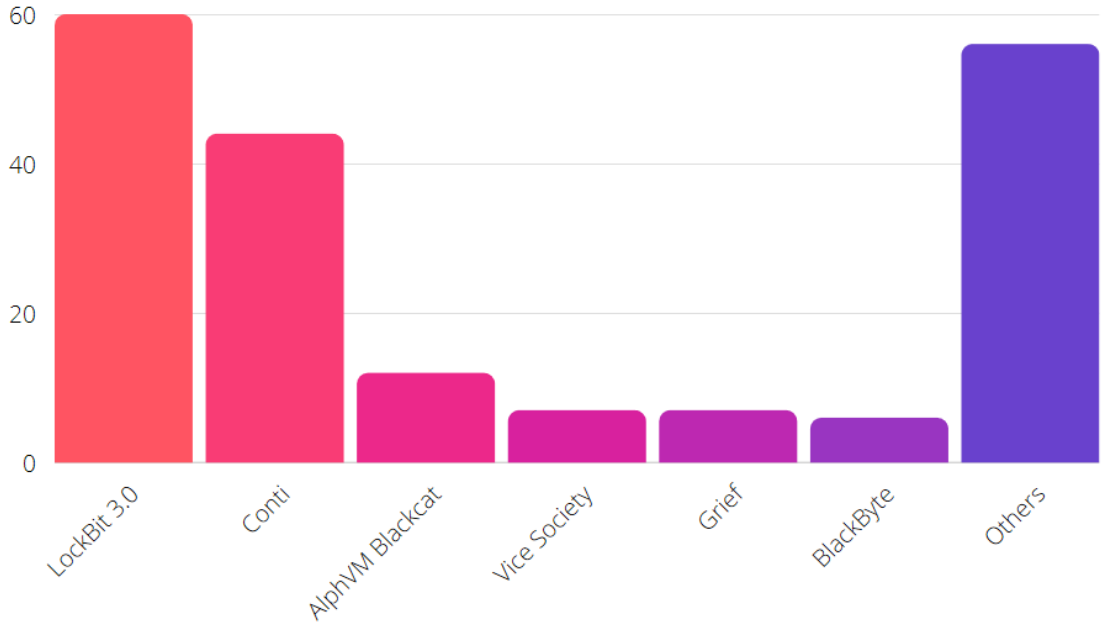


Figure 2.2: Most widespread ransomware in Italy in 2022, [5]

The last mention of this analysis goes to **Emotet**, which is without any doubt the most widespread malware in Italy. This malware uses the conflict in Ukraine as a vector of phishing campaigns, by sending malicious ZIP attachments inside emails. The aim of this malware is to steal banking information and credentials. Even if this is not a ransomware, it is important to notice the role of phishing, which can be the main vector not only for the previous type of malware, but for ransomware too.

Chapter 3

Important tools and protocols

As we have seen so far, ransomware are able not only to attack locally a system but in some cases, they are also able to propagate themselves over the network and connect and infect other systems. We need a way to detect this process, isolate the victim, and stop the ransomware from propagating as soon as possible. We want to take a look at the behavior of ransomware from the network point of view, so the focus will be on the SMB Protocol, which is so far one of the most used ones, especially in Microsoft Windows systems. Traces left by runs of attacks using this protocol will be then captured and analyzed using tools such as Wireshark and Zeek.

3.1 Zeek: an open source network security monitoring tool

Zeek is a passive and open-source network traffic analyzer. It is commonly used as a network security monitor to investigate over possible malicious activities. This tool can capture the traffic as packets that are sent to a queue where they will be then processed. Zeek is capable also to perform different traffic analysis tasks beyond the security domain, including performance

measurement and troubleshooting. Zeek is also fully customizable and extensible thanks to the Zeek Scripting language that let the user the possibility to write custom code and functionalities. [7]

3.1.1 Zeek Architecture

Zeek can sniff packets from the network and then send them to its first main component which is an *event engine* that reduces the incoming packet stream into a series of higher-level events. For example, an HTTP request corresponds to an *http_request* event that contains the involved IP addresses and ports, the requested URI, and the HTTP version. It does not convey any other interpretation (such as whether the URI corresponds to a known malware site). These events are then sent to the second main component called *script interpreter*, which executes some event handler using Zeek's scripting language. This allows the user to customize what will happen when an event is triggered. Scripts can derive any desired properties from the traffic and in fact, by using this tool, the first advantage that a user can experience is the high number of different sets of logs that describe the network activity. All these logs are generated thanks to the event handler. For example, if Zeek captures some SMB traffic, the related event will be triggered and a log file named *smb_files.log* will be generated. To better understand the architecture it is possible to look at Figure 3.1. Logs can be generated both from real-time traffic and already existing *.pcap*. In the first case, Zeek is always listening for new packets over the network, while the latter can be downloaded from any source or generated using Wireshark as we will see later. [7]

3.1.2 Installing and running Zeek

Zeek, which is open source, has been installed on a Windows machine using a Docker container. The information regarding the installation and first run is contained in Appendix A.1. The instructions will run the default configuration of Zeek that actually generates a set of logs based on sniffed traffic. These logs can be created in traditional CSV format but also JSON format by using a specific option while running Zeek. Since we want to

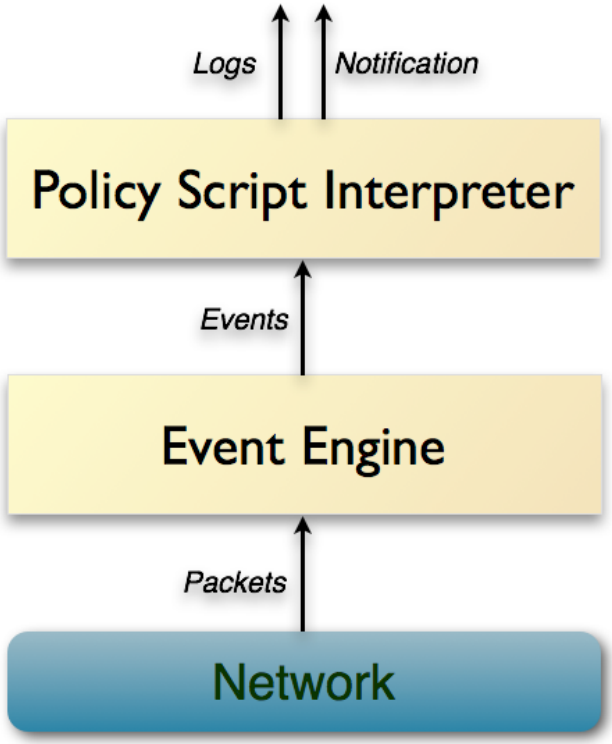


Figure 3.1: High level architecture of Zeek, [7]

customize this log generation (because we need some further parameters), a Zeek Script has been created for this purpose. [7]

3.1.3 Zeek Scripts

Zeek contains an event-driven scripting language that provides anything needed to extend and customize Zeek's functionalities. It is possible to consider Zeek as an entity that processes connections and generates events, while Zeek's scripting language is the medium through which it is possible to communicate. Scripts can be written on text files using *.zeek* extension. Even if it is a scripting language, it looks similar to a C-like language, but the official documentation provided by Zeek has been the core of this part of the job since it describes all the available types and structures and it

helped me a lot to learn how to write Zeek Scripts. To perform the requested ransomware detection, the generated logs must contain some specific SMB messages, must contain the MD5 of files (to alert about file modification, which will be discussed later), and must be in JSON format. The created zeek script is available in Appendix A.2.

3.1.4 Files and SMB Files logs

One of the main log files generated by Zeek that will be analyzed during the thesis is the *files.log* file. It contains records of files that Zeek observed while inspecting network traffic. This is important to understand what files passed on the network, more precisely which files have been modified. Some of the most important fields that each record contains in this file are reported in the following table:

Field	*Type*	*Description*
ts	double	packet timestamp
fuid	string	unique identifier of the file
tx_hosts	vector[string]	sender IP addresses
rx_hosts	vector[string]	receiver IP addresses
source	string	protocol that generated the connection
mime_type	string	identifies file formats
filename	string	name of the file
md5	string	md5 hash of the file

Table 3.1: Main parameters of files.log records

The next one is *smb_files.log*. This is the most important log file, since all the traffic that goes to the network that can refer to a SMB connection is saved here. The most important fields are reported in the following table:

3.1.5 Other log files

The previous log files are the ones used for the analysis of this thesis, but Zeek generates a variety of different log files, each one containing different

Field	*Type*	*Description*
ts	double	packet timestamp
uid	string	unique identifier of connection
id.orig_h	string	sender IP address
id.resp_h	string	receiver IP addresses
action	string	type of SMB message
name	string	file name

Table 3.2: Main parameters of smb_files.log records

information. Some of them are:

- **conn.log**: the connection log. Even if the file is named like this, Zeek tracks both TCP and UDP traffic.
- **dhcp.log**: DHCP logs help analysts map IP addresses to MAC addresses, and may also reveal hostnames;
- **dns.log**: this log records all the DNS requests.

Of course all the mentioned log files are generated only if that specific kind of traffic is sniffed from the network, otherwise the corresponding log file is not created.

3.2 SMB - Server Message Block Protocol

Server Message Block is a communication protocol that enables file sharing, printer sharing, network browsing, and inter-process communication (through named pipes) over a computer network. It was first developed by IBM but then widely implemented by Microsoft in its systems. SMB is a stateful protocol where clients open connections to the server, establish an authenticated context on that connection, and then issue various requests in order to access files, printers, and named pipes for inter-process communication, so it is clear that it relies on TCP and IP protocols for transport. [8] In order to establish connection and to exchange data, the protocol uses various messages which can be divided into the following categories:

Category	*Messages*
Protocol negotiation	SMB NEGOTIATE
User authentication	SMB SESSION_SETUP, SMB LOGOFF
Share access	MB TREE_CONNECT, SMB TREE_DISCONNECT
File access	SMB CREATE, SMB OPEN, SMB CLOSE, SMB READ, SMB WRITE, SMB FILE_RENAME
Directory access	SMB QUERY_DIRECTORY, SMB CHANGE_NOTIFY
Volume access	SMB QUERY_INFO, SMB SET_INFO
Cache coherency	SMB OPLOCK_BREAK
Simple messaging	SMB ECHO

Table 3.3: Categories of SMB messages, [8]

Ransomware need to perform read and write operations over files in order to be able to encrypt them. It is expected that the majority of entries of the protocol's log generated from the run of a ransomware attack will contain only a subset of the previous messages, more specifically the SMB READ and the SMB WRITE messages. The SMB READ Request packet is sent from the client to request the read operation on a file that will be specified in one of the fields, the FileId. The answer from the server to the request will be the SMB READ Response, that that will contain the variable-length buffer which contains the data read for the response. The SMB WRITE Request packet is sent from the client to write data to the file or named pipe on the server and it will contain the fileId and the variable-length buffer. The answer from the server, the SMB WRITE Response packet, will contain the number of bytes written. The high level of SMB architecture is depicted in Figure 3.2.

3.3 Wireshark and packet captures

Wireshark is an open source network packet analyzer. It tries to show captured packet data in as much detail as possible. It is basically possible to record what is happening over the network and to get a look at all the

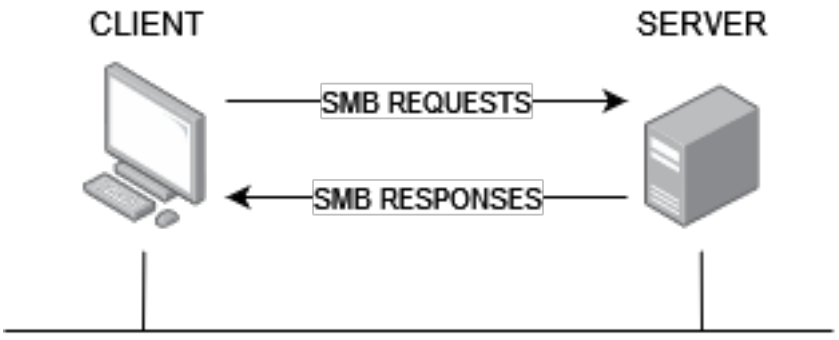
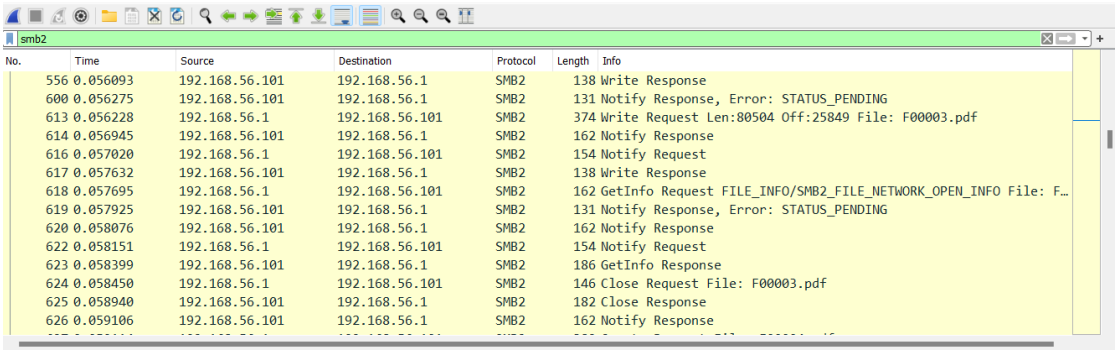


Figure 3.2: High level architecture of SMB

packets using a GUI which does provide filtering options too. The final results can be saved in a file using the *.pcap* format. For the purpose of this thesis, Wireshark needs to record live run of a ransomware scripts on target systems. In some cases, that will be shown later on, the tool has been used in order to record the SMB messages exchanged between client and server both during normal traffic exchange and under the attack of a ransomware. In the Figure 3.3 it is possible to look at a network capture filtered to show only SMB traffic.


smb2

No.	Time	Source	Destination	Protocol	Length	Info
556	0.056093	192.168.56.101	192.168.56.1	SMB2	138	Write Response
600	0.056275	192.168.56.101	192.168.56.1	SMB2	131	Notify Response, Error: STATUS_PENDING
613	0.056228	192.168.56.1	192.168.56.101	SMB2	374	Write Request Len:80504 Off:25849 File: F00003.pdf
614	0.056945	192.168.56.101	192.168.56.1	SMB2	162	Notify Response
616	0.057020	192.168.56.1	192.168.56.101	SMB2	154	Notify Request
617	0.057632	192.168.56.101	192.168.56.1	SMB2	138	Write Response
618	0.057695	192.168.56.1	192.168.56.101	SMB2	162	GetInfo Request FILE_INFO/SMB2_FILE_NETWORK_OPEN_INFO File: F…
619	0.057925	192.168.56.101	192.168.56.1	SMB2	131	Notify Response, Error: STATUS_PENDING
620	0.058076	192.168.56.101	192.168.56.1	SMB2	162	Notify Response
622	0.058151	192.168.56.1	192.168.56.101	SMB2	154	Notify Request
623	0.058399	192.168.56.101	192.168.56.1	SMB2	186	GetInfo Response
624	0.058450	192.168.56.1	192.168.56.101	SMB2	146	Close Request File: F00003.pdf
625	0.058940	192.168.56.101	192.168.56.1	SMB2	182	Close Response
626	0.059106	192.168.56.101	192.168.56.1	SMB2	162	Notify Response

Figure 3.3: SMB traffic network capture using Wireshark

Chapter 4

The state of art

To improve ransomware detection, it is first needed to understand how ransomware behave and how and what is currently detected from the network intrusion detection system developed by *aizoOn*. This system relies on a customized version of Zeek, which means that the detection is already performed by looking at Zeek's log files. This chapter will briefly describe the NIDS, and then will perform an analysis of some ransomware that are detected using this system.

4.1 NIDS developed by aizoOn

The proposed solution to the detection problem will be applied in an intrusion detection system developed by aizoOn that can perform monitoring of networks looking for threats. Its main characteristics are:

- automatic learning engine for the identification of anomalies;
- fast identification of malicious patterns;
- threat intelligence;
- cognitive visualization of data to provide immediate evidence of alarms.

The NIDS is composed of various blocks, each one with specific functionality. The first is the probe, based on Zeek, which is stateless and gets only real-time data, while data that is not real-time is sent to a central engine, which is server side. Finally, there is a central service that is out of the intranet and it executes updates and propagates information to the other parts. Ransomware detection is performed using the probe, by analyzing the traffic that will be sniffed using Zeek.

4.2 Common ransomware behaviors

As previously discussed, ransomware have similar behavior while performing attacks, but what actually characterizes one ransomware from the other is how the encryption of data is performed. In this step of the attack, tools like the one under consideration focus on some aspects, but less on others. It is possible to identify the behavior that ransomware can have while encrypting files in three categories:

- **Class A** ransomware overwrites the contents of files by opening it, reading its content, and then writing the encrypted content in place, and then closing the file. It can optionally rename the file.
- **Class B** ransomware extends Class A, with the addition that the file is moved out of the user's documents directory (e.g., into a temporary directory). Then malware reads the file, writes the encrypted payload, then moves the file back to the user's directory. Again, the file name may be different than the original file name.
- **Class C** ransomware reads the original file, then creates a new, independent file containing the encrypted contents and deletes or overwrites (using a move) the original file. [9]

While encrypting files, all classes can have again different behaviors:

- Ransomware encrypts some bytes at the beginning of a file, which usually corresponds to the header file. Any tool that analyses changes in these bytes can detect this kind of ransomware.

- Other ransomware let the first bytes of files untouched; thus, the tools that analyze the change of these bytes will not be able to detect them.

Having in mind the common behavior of ransomware, the next step is to analyze which behaviors are detected by the NIDS taken in exam.

4.2.1 Detection based on MIME types

The first approach for ransomware detection that is currently supported by the network intrusion detection system that this thesis will improve is the one based on MIME types. As previously introduced, some ransomware to make unreadable as much data as possible encrypts only some bytes of the header file, (also called magic bytes [4]), which is usually stored at the beginning of a file and contains metadata about the file and its content. This metadata is typically used by software to read and interpret the file during the loading process, but also afterward. Among the metadata, there is the MIME type which specifies the nature and format of a document and files. [10] It let the operating system understand what is the software in charge to handle that file. Since malware encrypts some bits of the header, the MIME-type is altered and software will not be able to understand how to process data and the file will become useless. In this way, malware needs to perform only quick operations on file to make it unreadable and this increases the speed of encryption and so the number of encrypted files. Regarding the detection performed by the NIDS, it starts from the consideration that there are basically two possibilities:

- **Overwrite** - happens if there is an SMB message with a read and then a write SMB message on two files with the same FUID and different MIME type;
- **Delete** - happens if there is a read SMB message and a delete SMB message on two files with the same file name and a write SMB message on a file with the same combination of source and destination IP and ports, different MIME type and similar file name.

Both ways are based on the detection of changes in the MIME type field and cover overwrite and delete of a file, but it is focused only on magic

bytes. In this way, any ransomware that does not perform any change to that field will not be detected by this approach. Of course, a change in the MIME type is not sufficient to understand that there is a ransomware attack going on, since a MIME type change can happen, for example, while upgrading the version of a Word document file from *.doc* (Microsoft Office Word versions up to 2003) *.docx* (currently used extension). For this reason, this indicator will be combined with others (for example the frequency of MIME-type modifications) to reduce the number of false positives detected by the software. Since this approach is already implemented, I need to find some ransomware that are able to bypass the NIDS, so I performed research for online resources about ransomware traffic captures.

4.3 Online resources

Research online does not provide a lot of information about ransomware creating network traffic while running. Some of the few repositories that have been found are private, which means that it is possible to get access to the resources only after payment, so I focused on the only free repository about traffic captures that I found on the web. The repository [4][11][12] comes from the Public University of Navarre (Spain) and it contains PCAP files obtained by executing ransomware binaries and capturing the network traffic created when encrypting a set of files shared from an SMB server. The aim is to provide ransomware traffic to test detection tools. This repository has been created by using a scenario typical of enterprise networks, in which user documents are centralized in file servers since this helps document sharing and backups. Of course, the used protocol is SMB. The basic parameters defining the scenario for each run of a ransomware sample are:

- **Operating system**: some of the run use SMB v2 protocol on Windows 7, while others use SMB v3 protocol on Windows 10;
- **Network speed**: it has been set to the maximum allowed (10 Gb/s);
- **Fileset**: the filesets are created randomly. This can deeply affect ransomware behaviour, since the fileset is the one encrypted by the ransomware sample.

I downloaded and analyzed two of these samples: WannaCry ransomware and Stop ransomware. The task here is to look at ransomware SMB traffic and in particular to the MD5 of files, since we want to detect a ransomware that performs the encryption of files by overwriting them. The MD5 of that file should be different: one value before the encryption, a different value after the encryption. We need to understand if this can be noticed by analyzing the samples.

4.3.1 Analysis of WannaCry ransomware traffic capture

The first analyzed sample is WannaCry ransomware. The downloaded .pcap has been directly opened using Zeek and a custom script using the instructions reported in Appendix A.2. The result of Zeek analysis is a set of logs. The interesting ones are *files.log* and *smb_files.log*. I decided to focus first on *smb_files.log* and more precisely on what a single file encrypted by the ransomware generates in terms of SMB traffic. Each encryption of a file generates 10 SMB different messages, and sorted by time they are reported in Table 4.1, which uses relative timestamps for simplicity. It is possible to

relative ts	**message**
1	SMB::FILE_OPEN
1	SMB::FILE_READ
1	SMB::FILE_READ
2	SMB::FILE_OPEN
2	SMB::FILE_WRITE
2	SMB::FILE_WRITE
3	SMB::FILE_OPEN
3	SMB::FILE_RENAME
4	SMB::FILE_OPEN
10	SMB::FILE_DELETE

Table 4.1: Messages generated for one file encryption by WannaCry

notice that more than a message is sent at the same time and that some of them occur multiple times. Any time an action on file is performed (read, write, rename) it is preceded by an open message that is encapsulated

together with the requested action, since they come at the same time. By looking at the first action, *SMB::FILE_READ*, we notice that it occurs two times and at the same time. The comparison between these two message that is reported on Table 4.2 shows that they are exactly the same message,

field	First READ content	Second READ content
ts	139.34302	139.34302
uid	CyT0pL347irAQkn4Jg	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4	192.168.1.4
id.orig_p	49191	49191
id.resp_h	192.168.1.5	192.168.1.5
id.resp_p	445	445
fuid	none	FyUpE32nMvwCleP5Vg
action	SMB::FILE_READ	SMB::FILE_READ
path	\\PCB \directorioCompartido	\\PCB \directorioCompartido
name	F10138.pdf	F10138.pdf
size	8270	8270
times.modified	1582207053.6709907	1582207053.6709907
times.accessed	1582207540.6734452	1582207540.6734452
times.created	1582207540.6734452	1582207540.6734452
times.changed	1582207540.6890702	1582207540.6890702

Table 4.2: Comparison between two READ messages happening at the same time

but the last one contains the *fuid* of the file, that will tell us which file is being read since the *fuid* will refer all file information in the other log file, *files.log*. After some research and after having contacted the developers of Zeek, I found out that it happens due to Zeek's analysis and they're not two real different read requests. Since they are perfectly the same, but the second one has more information, the first one is discarded and from now on it will not be shown anymore. The same happens also for write messages, they appear twice but only one is meaningful to us.

Let's now perform the analysis message by message of what happen to the file:

- **Timestamp 1**: *SMB::FILE_OPEN* and *SMB::FILE_READ* happen on file *F10138.pdf* that is the original and not encrypted user file. This is because the ransomware needs to read the file in order to perform encryption. In Table 4.3 it is possible to look in detail the fields of both messages.
- **Timestamp 2**: the *SMB::FILE_OPEN* is requested this time for file *F10138.pdf.WNCRYT* that does not exist on the target machine. This means that the file is created and will have size 0 and the next message, *SMB::FILE_WRITE*, will write the encrypted data on the newly created file. In Table 4.4 it is possible to look in detail the fields of both messages.
- **Timestamp 3**: after encryption is completed, the *SMB::FILE_OPEN* is requested on file *F10138.pdf.WNCRYT* (it is possible to notice here that size is no more 0 and size of encrypted file is higher than the size of the original one) and at the same time the *SMB::FILE_RENAME* modifies the name of file to *F10138.pdf.WNCRYT*. In Table 4.5 it is possible to look in detail the fields of both messages.
- **Timestamp 4**: an *SMB::FILE_OPEN* message for file *F10138.pdf.WNCRY* goes trough the network but this time no more actions happen. It is probably a check that file has been correctly renamed. In fact, size of file is the same of the previous *F10138.pdf.WNCRY*. In Table 4.6 it is possible to look in detail the fields of both messages.
- **Timestamp 10**: here the number of relative timestamp means that this action happens after a while with respect to all the other actions (have a look to the ts field). This is the last step in which the *SMB::FILE_DELETE* message ask for the deletion of *F10138.pdf* and at this point the original file is lost. In Table 4.7 it is possible to look in detail the fields of both messages.

Finally, in Table 4.8 there are the details for the two different fuid involved in the encryption of the file. It is important to notice two aspects that happen during the encryption of the file:

- **The fuid of original file and final encrypted file is different.** This happens because WannaCry after encrypting data creates a new

field	SMB::FILE_OPEN	SMB::FILE_READ
ts	139.34302	139.34302
uid	CyT0pL347irAQkn4Jg	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4	192.168.1.4
id.orig_p	49191	49191
id.resp_h	192.168.1.5	192.168.1.5
id.resp_p	445	445
fuid	none	FyUpE32nMvwCleP5Vg
action	SMB::FILE_OPEN	SMB::FILE_READ
path	\\PCB \directorioCompartido	\\PCB \directorioCompartido
name	F10138.pdf	F10138.pdf
size	8270	8270
times.modified	1582207053.6709907	1582207053.6709907
times.accessed	1582207540.6734452	1582207540.6734452
times.created	1582207540.6734452	1582207540.6734452
times.changed	1582207540.6890702	1582207540.6890702

Table 4.3: Focus on timestamp 1 actions of WannaCry samples

file where the encrypted content is stored and then the previous one is deleted. In this way, the original file is never modified except for deletion.

- **MIME type not available on encrypted file**. On the NIDS, Zeek is forced to recompute the MIME type anytime a file goes trough the network, but the fact that on encrypted file this filed is missing means that probably the ransomware is working on header files as we previously explained.

By sending this sample to the NIDS it detects that WannaCry is running and alerts about ransomware, as expected, but we cannot appreciate changes in md5, since files have always different fuids and we cannot rely on the filename to match the encrypted and original one, since filename can be completely altered.

field	SMB::FILE_OPEN	SMB::FILE_WRITE
ts	139.348404	139.348404
uid	CyT0pL347irAQkn4Jg	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4	192.168.1.4
id.orig_p	49191	49191
id.resp_h	192.168.1.5	192.168.1.5
id.resp_p	445	445
fuid	none	FNwqDo1QhJASIPmCj4
action	SMB::FILE_OPEN	SMB::FILE_WRITE
path	\\PCB \directorioCompartido	\\PCB \directorioCompartido
name	F10138.pdf.WNCRYT	F10138.pdf.WNCRYT
size	0	0
times.modified	1631867511.877	1631867511.877
times.accessed	1631867511.877	1631867511.877
times.created	1631867511.877	1631867511.877
times.changed	1631867511.877	1631867511.877

Table 4.4: Focus on timestamp 2 actions of WannaCry samples

4.3.2 Analysis of other ransomware traffic captures

Since the analysis on WannaCry reported that it does not perform any modification on the original file except for the delete, I analysed other samples to look if some of them work by encrypting the original file. For this reason, I performed the same analysis also on Stop ransomware. The number of generated SMB messages for one file encryption is the same, since they behave in a very similar way from the network point of view. The only interesting point that differ from WannaCry is that Stop creates a new file that has a different extension, but it also modifies the begin of the file name, as reported in Table 4.9 (original name was *F10583.pdf*). Again, since Stop performs MIME type change during encryption, by sending this sample to the NIDS it detects that Stop is running and alerts about ransomware as expected.

field	SMB::FILE_OPEN	SMB::FILE_RENAME
ts	139.362303	139.362303
uid	CyT0pL347irAQkn4Jg	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4	192.168.1.4
id.orig_p	49191	49191
id.resp_h	192.168.1.5	192.168.1.5
id.resp_p	445	445
fuid	none	FNwqDo1QhJASIPmCj4
action	SMB::FILE_OPEN	SMB::FILE_RENAME
path	\\PCB \directorioCompartido	\\PCB \directorioCompartido
name	F10138.pdf.WNCRYT	F10138.pdf.WNCRY
size	8552	8552
prev_name	not present	F10138.pdf.WNCRYT
times.modified	1582207053.6709907	1582207053.6709907
times.accessed	1582207540.6734452	1582207540.6734452
times.created	1582207540.6734452	1582207540.6734452
times.changed	1631867511.892625	1631867511.892625

Table 4.5: Focus on timestamp 3 actions of WannaCry samples

field	SMB::FILE_OPEN
ts	139.367597
uid	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4
id.orig_p	49191
id.resp_h	192.168.1.5
id.resp_p	445
action	SMB::FILE_OPEN
path	\\PCB \directorioCompartido
name	F10138.pdf.WNCRY
size	8552
times.modified	1582207053.6709907
times.accessed	1582207540.6734452
times.created	1582207540.6734452
times.changed	1631867511.892625

Table 4.6: Focus on timestamp 4 action of WannaCry samples

field	SMB::FILE_DELETE
ts	168.440041
uid	CyT0pL347irAQkn4Jg
id.orig_h	192.168.1.4
id.orig_p	49191
id.resp_h	192.168.1.5
id.resp_p	445
action	SMB::FILE_DELETE
path	\\PCB \directorioCompartido
name	F10138.pdf
size	8270
times.modified	1582207053.6709907
times.accessed	1582207540.6734452
times.created	1582207540.6734452
times.changed	1582207540.6890702

Table 4.7: Focus on timestamp 10 action of WannaCry samples

field	F10138.pdf	F10138.pdf.WNCRYT
ts	139.347709	139.350289
fuid	FyUpE32nMvwCleP5Vg	FNwqDo1QhJASIPmCj4
tx_hosts	192.168.1.5	192.168.1.4
rx_hosts	192.168.1.4	192.168.1.5
conn_uids	CyT0pL347irAQkn4Jg	CyT0pL347irAQkn4Jg
source	SMB	SMB
depth	0	0
mime_type	application/pdf	not present
filename	F10138.pdf	F10138.pdf.WNCRYT
duration	0.007588	0.005645
is_orig	false	true
seen_bytes	8270	8552
total_bytes	8270	not present
missing_bytes	0	0
overflow_bytes	0	0
timedout	false	false

Table 4.8: Focus on files.log entries for a file in WannaCry samples

field	SMB::FILE_OPEN	SMB::FILE_WRITE
ts	96.626004	96.626004
uid	CCEYkt346jBBElYBRk	CCEYkt346jBBElYBRk
id.orig_h	192.168.1.4	192.168.1.4
id.orig_p	49245	49245
id.resp_h	192.168.1.5	192.168.1.5
id.resp_p	445	445
fuid	none	FNwqDo1QhJASIPmCj4
action	SMB::FILE_OPEN	SMB::FILE_WRITE
path	\\\\192.168.1.5\\Users	\\\\192.168.1.5\\Users
name	[..] \\S10583.cqs.XAXAX0X0	[..] \\S10583.cqs.XAXAX0X0
size	0	0
times.modified	1574986149.483875	1574986149.483875
times.accessed	1574986149.483875	1574986149.483875
times.created	1574986149.483875	1574986149.483875
times.changed	1574986149.483875	1574986149.483875

Table 4.9: Focus on file creation of Stop samples

Chapter 5

Ransomware design and development

Online resources about pcap samples didn't provide any capture that can bypass the network intrusion detection system, since the behavior of recorded samples is really similar between one and another and they are all recognized. For this reason, I needed to design and implement a ransomware that does not alter in any way the header file but simply performs the encryption of the content without creating or deleting any file. This behavior is completely different from the one analyzed and in this way it could be possible to test if detection applies also in this case.

5.1 Ransomware design

As we mentioned, ransomware that encrypts some bytes of header files are not the target. Since no ransomware found on the web matches the needs, I developed a ransomware with different characteristics:

- **Content encryption:** ransomware has to encrypt only the content of files, without touching in any way the header file. This is required because in case the MIME type is modified, then it is correctly detected

and this is not wanted;

- **SMB traffic generation:** the ransomware has to work fine also on shared folders using SMB since we want to record the traffic and later perform the analysis.
- **No file creation:** ransomware must not create new files but has to overwrite the existing ones so that we can use MD5 checksum to understand that the same file has been modified.
- **Fast development:** since it is only for testing purposes, ransomware can be developed using Python, which provides fast development and good performance.

These characteristics are present in other ransomware as well, but the fact that online traffic capture samples are not available and downloading a real ransomware is dangerous, we decided to develop our ransomware. In this way, we have full control over what is happening in the system and we can recover fast in case of problems.

5.2 Ransomware development

The selected programming language used for development is Python. The script is composed of one class named *Ransomware* that contains three main functions:

- **Constructor:** in this function the starting root path for encryption is selected. Of course, this will be later substituted with the path of the shared folder, enabling in this way the generation of SMB traffic. In this function, there is also the generation of the symmetric key that will be used to encrypt files. We use a symmetric key because it is much faster than an asymmetric key and it is used in real applications for file encryption. It is possible (and highly suggested) to encrypt the symmetric key using an asymmetric key for security reasons, but in the script, for simplicity, it is written in clear (using the base64 encoder) on a file.

- **Encrypt_root_folder():** here the os library is used to walk through the file system starting from the root path previously selected. This function will iterate on each file contained in the directories of the tree and will call the next main function, *encrypt_file()*.

- **Encrypt_file():** this function opens the file in the directory as binary, gets the size of the file, and selects the last 70% of the content that will be encrypted. The remaining 30% will be left untouched, and since the encryption begins from the end of files, the part left untouched will be probably the header of the file and some of the content. In this way, we can be pretty sure that for the majority of the files, only the content will be encrypted. The encryption is performed by overwriting the content of the file, as wanted.

To better understand how the script is composed it is possible to look for the class diagram in Figure 5.1 and Appendix A.3 for the full script code.

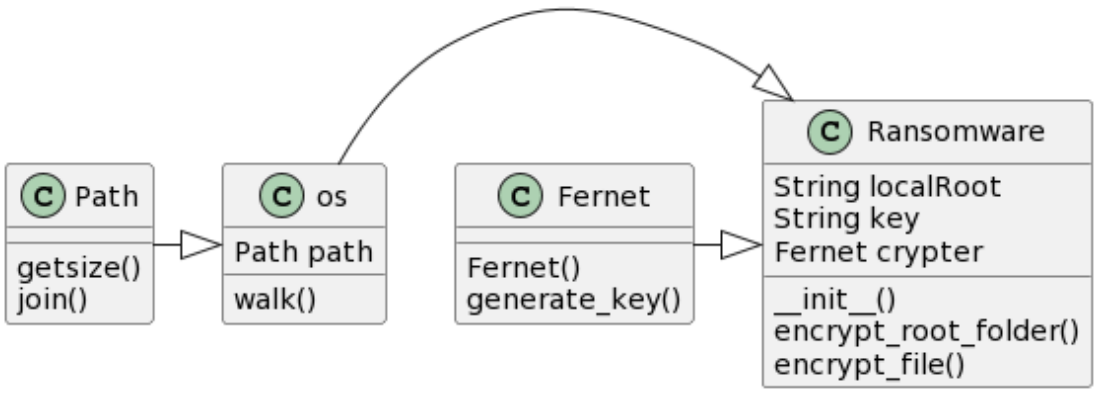


Figure 5.1: Ransomware class diagram

5.3 Setup of the environment and first run

After that script has been tested and it is fully working on the local machine, it is now needed to setup the environment to record the traffic that ransomware will generate while performing encryption on shared folders. We will use

again the SMB protocol and we will setup the SMB Server on a Linux virtual machine, that will be also used as a target system for file encryption.

The first step is to create the virtual machine. I used *Oracle VM VirtualBox* which is open-source software for the execution of virtual machines. Since it fully supports Linux, the selected image is *Ubuntu 22.04*. Everything has been downloaded from the official websites. After having configured the virtual machine to correctly run on my Windows system by giving the VM some RAM and virtual space, it is possible to proceed to the second step.

Second step is the configuration of the SMB server on a Linux machine, which has been performed following the instructions in Appendix A.4. In this step, since the folder to be shared is created, there is also the creation of fake files that will be encrypted. The used file extensions that are present in this shared folder are: *.txt*, *.docx*, *.pdf*, *.jpg*. They are mainly random files created ad-hoc or downloaded from the internet.

The final step is the part in which we run the ransomware to encrypt the shared folder (e.g., *sambashare*) and we also capture the traffic using Wireshark. So, we first verified that the shared folder is visible from the Windows machine, and then we setup correctly the root path that will be encrypted inside the variable of the ransomware script. In this case, it was *\\ip-address\sambashare*. It is possible to look at Figure 5.2, which describes a screen from Wireshark running on a Windows machine, to see that both hosts can communicate correctly using SMB, and that ransomware is starting looking for files in the root directory.

5.4 Analysis of generated traffic capture

As we did with other ransomware, we will perform now the analysis of the .pcap that has been created using Wireshark during the capture of the custom ransomware-generated traffic. As usual, the .pcap is first analyzed using Zeek and the custom script at Appendix A.2. Again, the logs taken in the analysis are *files.log* and *smb_files.log*. The analysis starts by focusing on the number of SMB messages generated by single file encryption, reported in Table 5.1. By comparing the table with the one generated by WannaCry at Table 4.1 it is possible to notice that this time there is no *SMB::FILE_DELETE* or

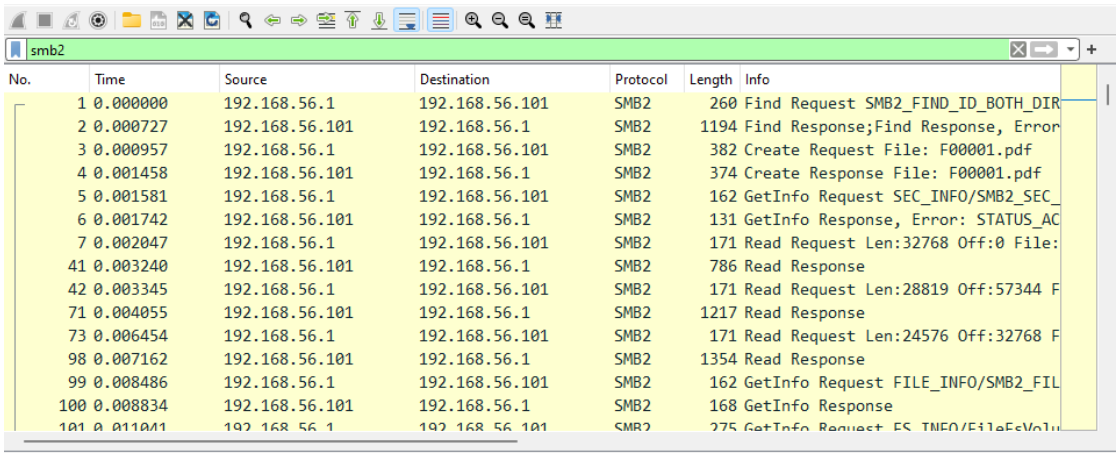


Figure 5.2: Recording ransomware SMB traffic using Wireshark

SMB::FILE_RENAME. Read and write operations are preceded as usual by an *SMB::FILE_OPEN* and the READ and WRITE messages are repeated twice, but only one containing the *fuid* as explained in Table 4.2. It is possible to deduce that this kind of ransomware generates fewer SMB messages than previous ones.

relative ts	message
1	SMB::FILE_OPEN
1	SMB::FILE_READ
1	SMB::FILE_READ
2	SMB::FILE_OPEN
3	SMB::FILE_OPEN
3	SMB::FILE_WRITE
3	SMB::FILE_WRITE

Table 5.1: Messages generated for one file encryption by custom ransomware

Let's perform the analysis on each timestamp to understand what happens to the file:

- **Timestamp 1**: At first, the original unencrypted file is opened and then read. The target host, that is the Linux VM, is the host with IP Address *192.168.56.101*. In Table 5.2 it is possible to look in detail the

fields of both messages.

- **Timestamp 2**: In this timestamp file is just opened. This happens because the ransom will first open again the file ready for writing the encrypted content, and then it starts writing. In Table 5.3 it is possible to look in detail the fields of both messages.

- **Timestamp 3**: Ransomware is encrypting by rewriting completely the content of file, which has been wiped from the previous content (in fact, size is 0). In Table 5.4 it is possible to look in details the fields of open and write messages.

field	**SMB::FILE_OPEN**	**SMB::FILE_READ**
ts	1666097601.635506	1666097601.635506
uid	CDqdzL33OKh93yyGg9	CDqdzL33OKh93yyGg9
id.orig_h	192.168.56.1	192.168.56.1
id.orig_p	53595	53595
id.resp_h	192.168.56.101	192.168.56.101
id.resp_p	445	445
fuid	not present	FetawXFHQopBoR7rj
action	SMB::FILE_OPEN	SMB::FILE_READ
name	F00002.pdf	F00002.pdf
size	86163	86163
times.modified	1666096258.9311624	1666096258.9311624
times.accessed	1666097571.1210535	1666097571.1210535
times.created	1666097571.1210535	1666097571.1210535
times.changed	1666096258.9311624	1666096258.9311624

Table 5.2: Focus on timestamp 1 actions of custom ransomware sample

The behavior is as expected and, in this case, we can also give a look to *files.log* fields about the same file since this time something interesting might come out, since file is not deleted but edited and there will be more than just one entry on the files' log. This was exactly what we wanted to achieve in order to understand the behavior of this kind of ransomware.

By looking at Table 5.5 we can immediately see that the encrypted file taken in exam appears two times in the log with the same *fuid*, of course in two

field	SMB::FILE_OPEN
ts	1666097601.646381
uid	CDqdzL33OKh93yyGg9
id.orig_h	192.168.56.1
id.orig_p	53595
id.resp_h	192.168.56.101
id.resp_p	445
action	SMB::FILE_OPEN
name	F00002.pdf
size	86163
times.modified	1666096258.9311624
times.accessed	1666097601.3641675
times.created	1666097571.1210535
times.changed	1666096258.9311624

Table 5.3: Focus on timestamp 2 actions of custom ransomware sample

field	SMB::FILE_OPEN	SMB::FILE_WRITE
ts	1666097601.649391	1666097601.649391
uid	CDqdzL33OKh93yyGg9	CDqdzL33OKh93yyGg9
id.orig_h	192.168.56.1	192.168.56.1
id.orig_p	53595	53595
id.resp_h	192.168.56.101	192.168.56.101
id.resp_p	445	445
fuid	not present	FetawXFHQopBoR7rj
action	SMB::FILE_OPEN	SMB::FILE_WRITE
name	F00002.pdf	F00002.pdf
size	0	0
times.modified	1666096258.9311624	1666096258.9311624
times.accessed	1666097601.3641675	1666097601.3641675
times.created	1666097571.1210535	1666097571.1210535
times.changed	1666096258.9311624	1666096258.9311624

Table 5.4: Focus on timestamp 3 actions of custom ransomware sample

different timestamp. The first one is recorded when original and unencrypted file is opened, the second one when file is modified and encrypted. In the

first entry it is the victim that gives the file to the ransomware, as it is possible to see in *tx_hosts* and *rx_hosts* fields, while in the second case it is the ransomware that writes on the victim machine the encrypted file. We also notice that MIME type remains **unchanged**, and this is exactly what we wanted: a ransomware that encrypts without altering the MIME type. It is also possible to notice the difference between original and encrypted file size by looking at *seen_bytes* field. The encrypted file has higher size than the original one. Finally, since file has been modified, it is possible to look at *md5* field to see that the hash changed due to the overwrite, so we notice the edit of the file. All the information extracted from these fields is

field	**First entry**	**Second entry**
ts	1666097601.637911	1666097601.650586
fuid	FetawXFHQopBoR7rj	FetawXFHQopBoR7rj
tx_hosts	192.168.56.101	192.168.56.1
rx_hosts	192.168.56.1	192.168.56.101
conn_uids	CDqdzL33OKh93yyGg9	CDqdzL33OKh93yyGg9
source	SMB	SMB
depth	0	0
analyzers	MD5	MD5
mime_type	application/pdf	application/pdf
filename	F00002.pdf	F00002.pdf
duration	0.004	0.0013
is_orig	false	true
seen_bytes	86163	106353
total_bytes	86163	not present
missing_bytes	0	0
overflow_bytes	0	0
timedout	false	false
md5	f17baca948e4fcae 5e9ac74dcd06fd39	63f7fc481d141e29 d9d9b13c406761a5

Table 5.5: Focus on files.log entries of custom ransomware sample

extremely precious because this is all that we have to perform the detection of this specific kind of ransomware. Furthermore, since MIME type is not altered in this case, the actual implementation of the intrusion detection system is not able to detect the ransomware. This is why the next chapter

will focus on the implementation of the detection.

Chapter 6

Detection

Now that there is a working ransomware and the traffic capture regarding it encrypting over the network using SMB protocol, it is now possible to implement the Java code that will be able to manipulate the previously analyzed JSON containing SMB messages to alert for dangerous operations happening on the network.

6.1 Solution design

The analysis of the SMB messages sniffed from the network allows us to think about a possible solution to the problem: how to detect that this ransomware is encrypting files? The answer is not simple, since in this case there is no possibility to rely on fields such as MIME type since it remains unchanged, and writing files remotely is not a strange behavior. The only possibility is to rely on the frequency of file edits, so on the number of SMB messages over time, and on the behavior of that user (for example, by keeping memory of the frequency of file edits of that user). We will also look for the md5 of files, since if it changes, then an edit happened. But first, some requirements are coming from the company, that will later use the code in its systems and will need to easily perform the integration. The main requirements are:

- **I/O independence**: code will be developed using static data saved on

file, but it will be later implemented on a system that uses Kafka, so it will need to manage real and live data. For this reason, the core of detection should be independent of the actual data received as input and sent as output.

- **Frequency analysis**: detection has to be performed using frequency analysis of read operations. There will be the computation of differences between one read and another divided by host and path, and then from these data, a threshold will be set for each host and path by using percentiles.

- **Spring framework**: since the real system relies on Spring, it is possible to use Spring framework and its functionalities as well on this code.

6.1.1 Code structure

The code has been developed in many steps, since in any step of the thesis I tested all the inputs by writing code. The starting point is the analysis of *files.log* to understand if it is possible to detect file edits by looking at md5. Later on, code has been integrated by looking at *smb_files.log* file to manipulate these data and divide SMB traffic by host and by paths. Then, together with *aizoOn* data scientists, we decided to compute the differences between two different SMB Read messages and to compute a threshold by using the mathematical concept of quantiles that are implemented in a class of Google Math library that has been used here. This threshold will be the time limit under which an alert will be sent as a potentially dangerous action. For reference, the class diagram is reported in Figure 6.1. We will now briefly explain the role of each class and the most important methods of the code.

Files and SmbFiles are the two classes with the aim of translating the JSON file into a Java object. In fact, it is possible to find here all the parameters that appear in a single action. Of course, not all possible parameters are reported here, but only those that may be helpful for the problem. Since the real implementation of Zeek on company systems may provide different names for JSON parameters, some of them are remapped to match their requirements. To do so, we use a Java Annotation on variables. An example is the following:

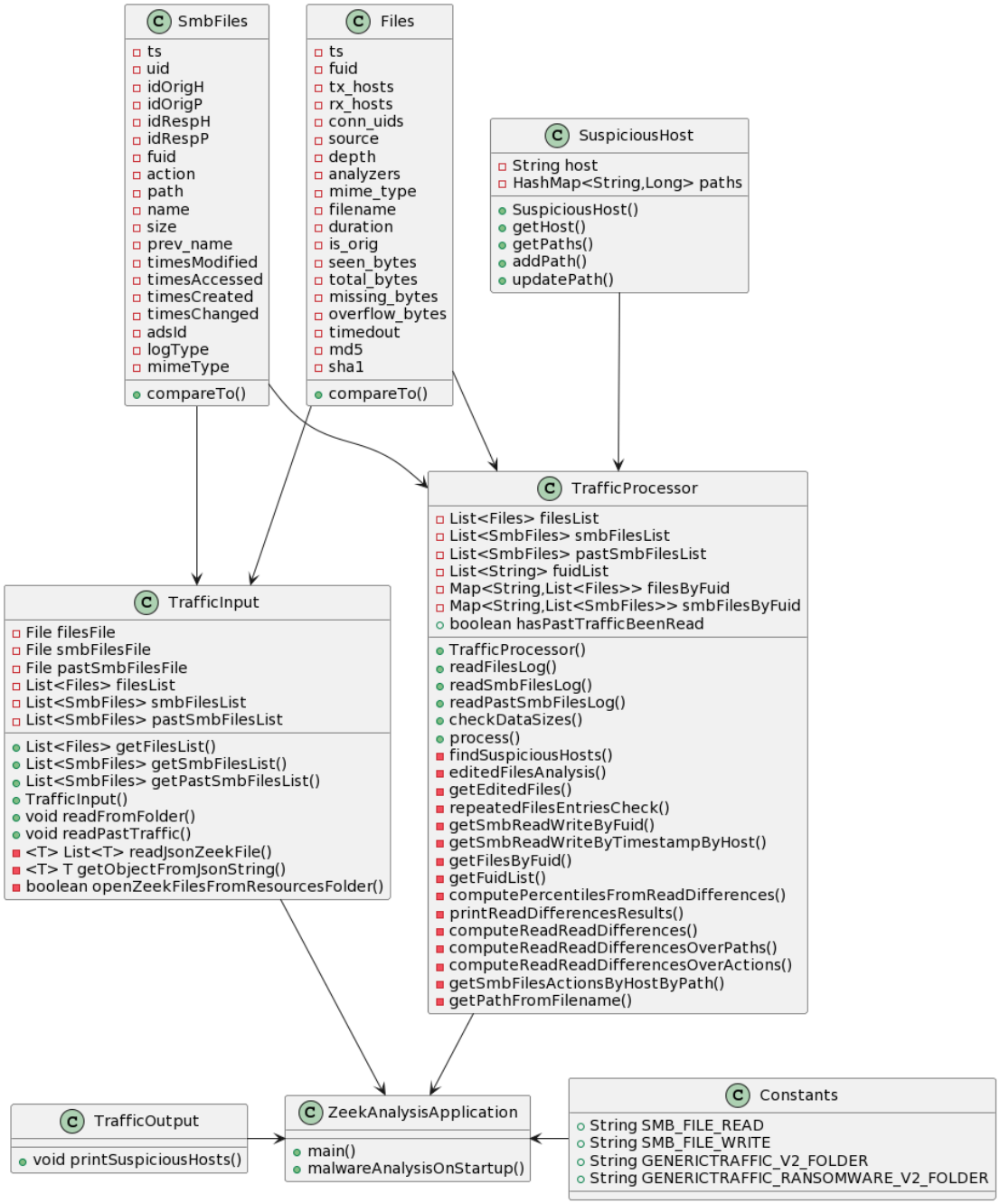


Figure 6.1: Class diagram of detection java code

```
1 @JsonProperty("times.changed")
```

```
@JsonAlias("times_changed")
private Double timesChanged;
```

In default Zeek configuration the *timesChanged* parameter is named *times.changed* but on real implementation it may be *times_changed*. To avoid problems during translation, we use the *JsonAlias* decorator. The Files and SmbFiles classes implement the *Comparable* interface to allow the override of *compareTo* for comparisons based on timestamps after the reading phase is completed. The implementation of the compareTo function is reported in the following lines in the form of pseudocode:

```
@Override
public int compareTo(Files that) {
    if(this.ts < that.ts) return -1;
    else if(this.ts == that.ts) return 0;
    else return 1;
}
```

TrafficInput is the class that, in this case, performs the reading of *files.log* and *smb_files.log* from the given folder. It reads JSON from files and creates the lists of corresponding java objects that will be available through the getter methods. It can easily be substituted with anything else that generates the same list of objects. One of the main methods here is the templated *readJsonZeekFile* function. It works in the following way:

```
private <T> List<T> readJsonZeekFile(File file, Class<T>
   valueType) {
    List<T> list = new List<>();
    try {
        BufferedReader br = new BufferedReader(new FileReader
   (file));
        while ((String line = br.readLine()) != null)
            list.add(getObjectFromJsonString(line, valueType)
   );
    } catch (IOException e)
        e.printStackTrace();
    return list;
}
```

It receives the file from which JSON needs to be read and the class object in which this JSON has to be translated. It simply scans the file by rows and calls *getObjectFromJsonString* function that actually performs the translation and is reported here:

```
private <T> T getObjectFromJsonString(String json, Class<T>
    valueType) throws JsonProcessingException {
     ObjectMapper mapper = new ObjectMapper();
     return mapper.readValue(line, valueType);
}
```

If the above function fails, it will throw an exception.

TrafficOutput is the class that will save the eventually generated alerts. In this case, this function will easily print on standard input all the alerts. No code is provided since it is a simple *println* function of result class parameters.

Constants is a class used to declare only constants. It contains definitions that are useful to switch the analysis on other traffic captures that are available in the resources folder of the project.

ZeekAnalysisApplication is the main class when a Spring project is created, since it contains the main function that will start the application. Here I added a method that will be executed after the *ApplicationReadyEvent* is emitted, basically on startup. This method will define the source taken as input (*TrafficInput*), and then will give the needed information for the analysis to the *TrafficProcessor* that is instanciated here.

```
@EventListener(ApplicationReadyEvent.class)
public void malwareAnalysisOnStartup() throws IOException {
    TrafficInput ti = new TrafficInput();
    ti.readJsonFiles();
    ti.readPastTraffic();
    TrafficProcessor tp = new TrafficProcessor();
    tp.readFilesLog(ti.getFilesList());
    tp.readSmbFilesLog(ti.getSmbFilesList());
    tp.readPastSmbFilesLog(ti.getPastSmbFilesList());
    tp.process();
```

```
}
```

It is possible to notice that we first read JSON files from a specific folder in this case, and then when the TrafficProcessor is instantiated it receives the corresponding Java objects, so it doesn't care about the source of these data. When everything is ready, the *process()* function is called.

TrafficProcessor this is the core class where the detection is performed. It receives as input the list of java objects of both *files.log* and *smb_files.log* on creation as we have seen before. To start the analysis, the method **process()** is called and it is reported in the following lines always in the form of pseudocode:

```
public void process() {
    if(hasPastTrafficBeenRead) proceed else return;
    if (checkDataSizes) proceed else return;
    if(areThereRepeatedEntriesInFiles(filesByFuid)) proceed
   else return;

    Map<String,List<SmbFiles>>
   pastSmbReadWriteByTimestampByHost =
   getSmbReadWriteByTimestampByHost(pastSmbFilesList);
    Map<String,List<SmbFiles>>
   newSmbReadWriteByTimestampByHost =
   getSmbReadWriteByTimestampByHost(smbFilesList);

    Map<String,Map<String, List<Double>>>
   pastTrafficReadDifferences = computeReadReadDifferences(
   pastSmbReadWriteByTimestampByHost);
    Map<String,Map<String, List<Double>>>
   newTrafficReadDifferences = computeReadReadDifferences(
   newSmbReadWriteByTimestampByHost);

    if(newTrafficReadDifferences.isEmpty() ||
   pastTrafficReadDifferences.isEmpty()) return;

    Map<String,Double> percentiles =
   computePercentilesFromReadDifferences(
   pastTrafficReadDifferences);

```

```
16      List<SuspiciousHost> suspiciousHosts =
        findSuspiciousHosts(percentiles, newTrafficReadDifferences
        );
17
18      editedFilesAnalysis(suspiciousHosts);
19
20      TrafficOutput.printSuspiciousHosts(suspiciousHosts);
21  }
```

At the beginning of the *process()* method some checks are performed in order to make sure that everything has been set up correctly for the reading of SMB traffic. The first step is to check that *files.log* contains some repeated entries for the same fuid. This is basically an md5 check on files, in order to understand if users made some changes to files. If there is none, then this kind of analysis cannot be performed since we are focusing on file changes.

The next step is performed two times because we need to use two kind of data traffic:

- **Past SMB traffic:** we will need to have past SMB traffic in order to understand the behavior of users and to compute the threshold that will be used as alert for the new incoming traffic.
- **New SMB traffic:** this new traffic will pass trough the threshold check in order to detect suspicious activities.

When *getSmbReadWriteByTimestampByHost()* method is called, it will start manipulation of data by performing some filtering actions. The result will be a timestamp-ordered map divided by host and containing only READ and WRITE messages, since that is all that we need. This method is called on both new and past SMB traffic. We use Java streams here since it is a powerful tool for data manipulation introduced in Java version 8 and it makes available many useful methods. It is possible to summarize the function with the following piece of pseudocode:

```
1  private Map<String,List<SmbFiles>>
       getSmbReadWriteByTimestampByHost(List<SmbFiles>
       smbFilesList) {
```

```
    return smbFilesList.stream()
        .filter(elements with IdOrigH and Fuid not null)
        .filter(elements that are SMB FILE READ or SMB FILE WRITE messages only)
        .sortByTimestamp()
        .groupBy(IdOrigH)
}
```

Next step is the computation of timestamp differences. This function is the one that looks, for each host and for each specific path, the differences between one read and another. At first we computed both differences between one read and another and for one write and another, but then we decided that the useful one is based on read differences, since normal traffic is not usually composed of many files read in short times and as we discussed before the ransomware needs to read files in order to compute the encryption.

```
private Map computeReadReadDifferences(Map<String,List<SmbFiles>> smbFilesByTimestampByHost) {
    Map result = new Map();

    if(smbFilesByTimestampByHost.isEmpty()) return;

    Map smbFilesActionsByHostByPath = getSmbFilesActionsByHostByPath(smbFilesByTimestampByHost);

    smbFilesActionsByHostByPath.forEach((host, smbFilesByPath) -> {
        Map differencesByPath = computeReadReadDifferencesOverPaths(smbFilesByPath);

        if(differencesByPath.size() > 0)
            result.put(host, differencesByPath);
    });

    return result;
}
```

This function receives the smbFilesList ordered by timestamp and divided by host. It then divide the traffic by path, so the result will be a map with host as key, and the value is again a map with path as key and action as value.

Then, it iterate over each host to compute differences over each path using *computeReadReadDifferencesOverPaths()* that will actually iterate over each path to compute the differences between READ actions. The result will be a map divided by host and path with corresponding list of difference values computed on that path. Again, this computation is performed on both past and new SMB traffic.

Next important step of *process()* method is the computation of percentiles starting from READ differences. It is computed by calling *computePercentilesFromReadDifferences()* method on past SMB traffic only. The tasks performed by this method are reported here, again in the form of pseudocode:

```
1  private Map<String,Double>
       computePercentilesFromReadDifferences(Map readDifferences)
        {
2       Map<String,Double> result = new Map();
3
4       readDifferences.forEach((host, differencesByPath) -> {
5           double[] valuesArray = differencesByPath.values().
       stream().mapToDouble(Double::doubleValue).toArray();
6           double quantile = percentiles().index(2).
       computeInPlace(valuesArray);
7           result.put(host,quantile);
8       });
9
10      return result;
11 }
```

The function iterates over READ differences computed before and, for each host, it computes the 2-percentile by using the mathematical function for percentiles computation. The method provided from Google Math Library will look for the distribution of all the differences and will take as threshold the shape of the tail, so the 2% most low value of all the differences values. In this way, each host will have a threshold set under which an alert will be provided. The value will highly influence the detection of false positives and that is why we decided to compute the 2-percentile. Of course, higher is the value, then higher will be the trehsold and higher the number of false positives.

These values will be the threshold for the analysis of the new incoming traffic, that starts with the method *findSuspiciousHosts()*. By looking at the new traffic, we count the number of differences that are lower than the percentile and, if counter is greater than 0 than it will be a suspicious host but still no alert is provided because we need ot perform one final step. The pseudocode of this function is reported here:

```
private List<SuspiciousHost> findSuspiciousHosts(Map
   percentiles, Map trafficReadDifferences) {
    List<SuspiciousHost> results;
    for(each entry in trafficReadDifferences.entrySet()) {
        String host = entry.getKey();
        Map differencesByPath = entry.getValue();

        for(each hostEntry in differencesByPath.entrySet()) {
            String path = hostEntry.getKey();
            List<Double> differences = hostEntry.getValue();

            double percentile = percentiles.get(host);
            long count = differences.stream()
            .filter(difference lower than percentile).count()

            if(count > 0) {
                add host to suspiciousHosts list
            }
        }
    }

    return results;
}
```

After looking for differences above the tresholds, the list of suspicious hosts and paths is created. The final step is to look for file changes in suspicious paths. If no file WRITE is performed, no suspicious activies are performed. For this reason, method *editedFilesAnalysis()* will get all the Write actions sniffed from the new traffic network, and by iterating on suspicious host previously computed will check if some of them are editing files in suspicious paths. If so, then an alert is sent and detection is performed.

SuspiciousHost is the class that is used in *process()* method to identify

suspicious hosts and how many suspicious activities happened on each path of that host.

6.1.2 Testing

Since the development of the code has been performed offline, the files used for testing are the ones generated by Zeek from online captures that have been analyzed previously and from the custom ransomware capture. By running the code on WannaCry sample, there is no result since they do not perform edits on existing files as we discussed. By running the code on the custom ransomware sample, alert is correctly provided. Of course, this is a "fake" environment, and for this reason the company gave me some sniffed traffic that has been generated by the working system on a real network, where there is no ransomware running. I splitted this traffic in two files, in order to simulate "past" and "new" traffic and performed the run of the analysis. Some alerts are provided, even if there is no suspicious traffic in it. This happens because, of course, this solution is not 100% precise and some **false positives** are possible. It is possible to play with the percentile parameter in order to establish different thresholds, but in any case some false positive may be reported. As a final step, I injected the ransomware traffic in the good one coming from the real traffic by editing the files manually, in order to understand if the real ransomware is alerted too and the result is achieved together with a couple of false positives.

Chapter 7

Conclusion

During the thesis, I analyzed in detail the SMB messages generated by ransomware running on the network to understand if there is a way to detect some specific kind of ransomware. The result is that it is possible, but how detection can be performed highly depends on what the ransomware does during encryption. In fact, I needed to create a specific kind of ransomware to meet the requirements and finally developed a code able to detect it.

The adopted technique, based on frequency analysis, is of course a statistical method that might fail and, as previously mentioned, it can lead to **false positives**. This is a consequence of the fact that the time in which an action happens is not certainly suspicious, as for example, it can be if the MIME type of the same file unexpectedly changes. Of course, it is possible to make the analysis more precise by better defining the threshold and so, by computing a different percentile on the differences of read actions.

The good news is that the method works since this specific kind of ransomware is detected and reported as it should be. But, of course, it is possible to have better results by reducing the number of false positives.

7.1 Possible evolutions of the code

The developed code actually works only on static files saved on disk and it only prints alerts on standard input. Of course, during the integration of the network intrusion detection system, code must be reviewed to make it work with a real chunk of data coming from Kafka topics and no more from static files.

It is also possible to modify how the threshold is computed in order to reduce the number of false positives, but it is also possible to integrate this threshold with the number of file edits that a host performs. Usually, read operations are much more frequent than write operations when a user has good behavior, while ransomware needs to perform lots of write operations. The idea is to count the number of file edits and to set a threshold, based on statistical analysis too, also for the number of this kind of action. By combining the frequency analysis with the number of file edits it is definitely possible to reduce the number of false positives.

To better reduce the number of false positives, it could be possible to use this detection method together with another tool that can look at all the running processes of host machines. In this way, if we notice for example that the WRITE action of a *.docx* file is performed from a Microsoft Office Word process, then it is for sure not a suspicious activity. Different is the result if the WRITE SMB messages are generated from an unknown process (e.g., a ransomware). In this way, together with the detection performed during this thesis, it will be highly probable that the activity is suspicious and that the number of false positives can be highly reduced.

Appendix A

Appendix A

A.1 Instructions for Zeek installation

Zeek comes out with a default configuration by providing transaction data and extracted content data, in the form of logs summarizing protocols and files seen traversing the wire. This configuration of Zeek has been installed using the official Docker image with the following command:

```
$ docker pull zeek/zeek:latest
```

As an example, the command to perform the analysis over a pcap file using the default Zeek configuration running on a Docker container is the following:

```
$ docker run --rm -v /path/to/pcap:/pcap:rw zeek/zeek zeek
 -r example.pcap local
```

- *docker run* is the command used to run the specified image, in this case zeek/zeek, in isolated containers;
- *–rm* is the option to automatically remove the anonymous volumes

associated with the container when the container is removed;

- *-v* is the option to create the volume. The first parameter is the path where the pcap file is stored (e.g., /path/to/pcap), then the path of the folder inside the volume (e.g., /pcap) and finally the "rw" option in order to make the volume readable and writable;
- *zeek -r* is the option to provide the pcap file as input of the Zeek process that will start once the container will be created.

The result of the Zeek run over the pcap file, that is a set of logs, will be saved in the path where the pcap file is stored in the computer (e.g., path/to/pcap).

A.2 Zeek custom script

The created Zeek Script that match the need of this thesis is the following:

```
@load base/protocols/smb/

export {
    redef SMB::logged_file_actions = set(SMB::FILE_READ, SMB
   ::FILE_WRITE,
        SMB::PRINT_CLOSE, SMB::FILE_DELETE, SMB::FILE_OPEN,
   SMB::FILE_RENAME,
        SMB::PRINT_OPEN);
    redef LogAscii::use_json = T;
}

event zeek_init() {
    print "Starting pcap analysis...";
}

event file_sniff(f: fa_file, meta: fa_metadata) {
    Files::add_analyzer(f, Files::ANALYZER_MD5);
}

event zeek_done() {
```

```
19     print "Pcap analysis completed!";
20 }
```

- The *export* block contains some declarations that the current module is exporting. This block enables global identifiers to be visible in other modules via the namespace operator (::). In this way, it is possible to use *redef* command in order to redefine some identifiers.

- *SMB::logged_file_actions* contains the file actions which are logged, that by default does not print SMB READ and SMB WRITE actions. For this reason, it is redefined.

- *LogAscii::use_json* is a boolean variable, false by default, to decide if logs must be printed in JSON format or not. We set it to true.

- *file_sniff* is an event that provide all metadata that has been inferred about a particular file from inspection of the initial content that has been seen at the beginning of the file. It is wanted to add here the MD5 analyzer, in order to get the md5 of each file inside logs.

- *zeek_init and zeek_done* events happen respectively at the beginning and at the end of the analysis. It is used only to print messages on console.

To run Zeek using Docker and a custom Zeek Script, the command to be used is the following:

```
1  $ docker run --rm -v /path/to/pcap:/pcap:rw zeek/zeek zeek
    -r example.pcap script.zeek local
```

Which is the same as the previous one, with the explicit declaration of the Zeek Script file.

A.3 Custom ransomware script

The following is the script that encrypts the content of any file given a specific root. The encryption starts from the end of a file up to the header but only for the 70% of the content. The remaining 30% remains unencrypted in order to avoid changes in the MIME type since we want a ransomware that does not alter the header but overwrites the content. Since it is only for testing purpose, the generated symmetric key for encryption is saved in clear in the same folder of the script, encoded in base64.

```
# Library to encrypt/decrypt files on target system
from cryptography.fernet import Fernet
# Library to get access on file system
import os

class Ransomware:
    def __init__(self):
        # Sysroot creates absolute path for files, etc. to encrypt the whole system
        # self.sysRoot = os.path.expanduser('~')
        # but for testing, use localroot
        self.localRoot = r'/home/user/ransomware-test/localRoot'

        # Generates a url safe(base64 encoded) symmetric key
        self.key =  Fernet.generate_key()
        # Save key on file
        f = open("symm.key", "w")
        f.write(self.key)
        f.close()
        # Creates a Fernet object with encrypt/decrypt methods to encipher data
        self.crypter = Fernet(self.key)

    def encrypt_file(self, file_path, encrypted=False):
        # Open file
        with open(file_path, 'rb') as f:
            # Read file size
            file_size = os.path.getsize(file_path)

```

```
28              # Compute the size of that file that will be
    encrypted
29              encr_size = int(file_size * 0.7)
30
31              unencrypted_data = f.read(file_size-encr_size)
32
33              # Encipher the last 70% of file
34              f.seek(-encr_size,2)
35              data = f.read()
36              if not encrypted:
37                  # Encrypt data from file
38                  _data = self.crypter.encrypt(data)
39
40          with open(file_path, 'wb') as fp:
41              # Write encrypted/decrypted data to file using
    same filename to overwrite original file
42              fp.write(unencrypted_data)
43              fp.write(_data)
44
45      def encrypt_root_folder(self, encrypted=False):
46          # Start encrypting from here
47          startingTree = os.walk(self.localRoot, topdown=True)
48          # Iterate over each folder in the tree looking for
    files
49          for root, dir, files in startingTree:
50              for file in files:
51                  file_path = os.path.join(root, file)
52
53                  if not encrypted:
54                      # encrypt file
55                      self.encrypt_file(file_path)
56                  else:
57                      # Skip
58                      self.encrypt_file(file_path, encrypted=
    True)
59
60  def main():
61      ransomware = Ransomware()
62      # Start encryption
63      ransomware.encrypt_root_folder()
64      print('Encryption complete.')
65
66  if __name__ == '__main__':
67      main()
```

A.4 Configuration of SMB server on Linux systems

In order to be able to share folders from Linux to the local network, it is needed to install and configure an SMB Server. The steps followed are reported below.

At first we need to install the SMB Server and the SMB filesystem. To do so, we used the following command:

```
sudo apt-get install samba smbfs
```

After having installed SMB, the next step is to create the folder that will be shared (and later encrypted remotely). We can do the following:

```
mkdir /home/user/sambashare/
```

Then, we need to open the SMB configuration file that is located in */etc/samba/smb.conf* using an editor, such as nano.

```
sudo nano /etc/samba/smb.conf
```

At the bottom of the file we need to provide the folder that we want to share. The correct way is the following:

```
[sambashare]
    comment = Samba on Ubuntu
    path = /home/user/sambashare
    read only = no
    browsable = yes
```

The parameters provided to the SMB configuration file are:

- *comment*: A brief description;
- *path*: the directory of the folder to be shared;
- *read only*: permission to modify the contents of the shared folder;
- *browsable*: when *yes*, file managers will list this shared folder under "Network".

After that we save the configuration file, we need to restart the SMB service and to update firewall rules in order to allow SMB traffic:

```
sudo service smbd restart
sudo ufw allow samba
```

As a final step, since SMB does not use the system account password, we need to setup a specific SMB password for the user account, using the following command:

```
sudo smbpasswd -a user
```

Now, it is all set. To access remotely to the shared folder, for example from a Windows machine, it is just needed to open the File Manager and to go to the following path:

```
\\ip-address\sambashare
```

Where the *ip-address* is the IP address of the SMB server machine (that has to be on the same network) and *sambashare* is the shared folder. [13]

Bibliography

[1] National Institute of Standards and Technology. *Protecting Information and System Integrity in Industrial Control System Environments: Cybersecurity for the Manufacturing Sector*. 2022. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-10.pdf.

[2] Ilker Kara and Murat Aydos. «The rise of ransomware: Forensic analysis for windows based ransomware attacks». In: *Expert Systems with Applications* 190 (2022). DOI: 10.1016/j.eswa.2021.116198. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85120993235&doi=10.1016%5C%2fj.eswa.2021.116198&partnerID=40&md5=44d0c757cd9e72c4d2b479134ef75877.

[3] Hon Lau Kevin Savage Peter Coogan. *The evolution of ransomware*. 2015. URL: https://docs.broadcom.com/doc/the-evolution-of-ransomware-15-en.

[4] Eduardo Berrueta, Daniel Morato, Eduardo Magaña, and Mikel Izal. «Open Repository for the Evaluation of Ransomware Detection Tools». In: *IEEE Access* 8 (2020), pp. 65658–65669. DOI: 10.1109/ACCESS.2020.2984187.

[5] SOCRadar. *Italy Threat Landscape Report*. 2022. URL: https://socradar.io/wp-content/uploads/2022/09/Italy-Threat-Landscape-Report.pdf.

[6] Puja Mahendru. *The State of Ransomware in Manufacturing and Production 2022*. 2022. URL: https://news.sophos.com/en-us/2022/10/26/the-state-of-ransomware-in-manufacturing-and-production-2022/.

[7] The Zeek Project. *Zeek Documentation*. 2023. URL: https://docs.zeek.org/en/master/.

[8] Microsoft Corporation. *[MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3*. 2022. URL: https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB2/%5bMS-SMB2%5d.pdf.

[9] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin R. B. Butler. «CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data». In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 2016, pp. 303–312. DOI: 10.1109/ICDCS.2016.46.

[10] Ned Freed, Dr. John C. Klensin, and Tony Hansen. *Media Type Specifications and Registration Procedures*. RFC 6838. Jan. 2013. DOI: 10.17487/RFC6838. URL: https://www.rfc-editor.org/info/rfc6838.

[11] Eduardo Berrueta, Daniel Morató, Eduardo Magaña, and Mikel Izal. «Open Repository for the Evaluation of Ransomware Detection Tools». In: (2020). DOI: 10.21227/qnyn-q136. URL: https://dx.doi.org/10.21227/qnyn-q136.

[12] Ransomware PCAP repository. *Eduardo Berrueta*. URL: http://dataset.tlm.unavarra.es/ransomware/.

[13] Aden Padilla. *Install and Configure Samba*. URL: https://ubuntu.com/tutorials/install-and-configure-samba.