Ransomware detection and mitigation tool

Jesper B. S. Christensen
Niels Beuschau

Kongens Lyngby 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

In computer science, ransomware is a field in constant development. Since antivirus and detection methods are constantly improved in order to detect and mitigate ransomware, the ransomware itself becomes equally better to avoid detection. Several new methods are implemented and tested in order to optimize the protection against ransomware on a regular basis.

The primary goal of this thesis is to create a tool able to detect and mitigate live ransomware. This ransomware already has infected the windows 10 system that this thesis tests upon. This tool will contain different methods of detection in order to identify a ransomware attack the fastest and stop that attack. The purpose of the created tool is neither to be an antivirus nor as robust as one, but solely to be a tool to detect and mitigate ransomware.

Since ransomware is a malware, to test it upon a system is a substantial thing to do, especially when doing many tests. Therefore all ransomwares are tested upon virtual machines, this means that all types of ransomware that has anti simulation methods and does not encrypt files when registering that it is a virtual machine, will not be tested in this thesis.

The different variants for the detection methods made, have been tested with 65 different ransomwares. The results for these variants has been found and analyzed and the ransomwares that the detection methods were tested upon has been analyzed as well. The result of this thesis is a solution that is able to detect active ransomwares and after a short delay stop the encryption process, thus stopping the active ransomware in 77% of all cases.

Summary (Danish)

Ransomware er et felt indenfor informationsteknologi, der stadig er i rivende udvikling. Eftersom antivirus og detekterings metoder konstant bliver forbedret i at opdage ransomware, bliver ransomware tilsvarende bedre til at undgå opdagelse. Mange nye metoder bliver stadig afprøvet for at optimere beskyttelsen mod ransomware.

Målet for denne afhandling er at skabe et værktøj der kan opdage og standse aktiv ransomware, der i forvejen har inficeret et windows 10 system, som denne afhandling tester på. Dette værktøj bliver bygget på forskellige detekterings metoder for hurtigst at opdage aktiv ransomware og standse det. Meningen med værktøjet er ikke at det skal være en antivirus eller ligeså robust som en, men derimod udelukkende et værktøj til detektering og begrænsning af ransomware.

Eftersom at teste ransomware er omfattende i forhold til testmiljø, da det er en virus, bliver alle tests med aktiv ransomware testet på virtuelle maskiner, derfor bliver ransomware der ikke er aktive på virtuelle maskiner ikke testet i denne afhandling.

Varianterne af detekteringsmetoderne er blevet testet mod 65 forskellige aktive ransomwares. Resultaterne for disse varianter er blevet sat op og analyseret og de ransomwares som methoderne blev testet på er også blevet analyseret. Resultatet er et produkt der kan detektere ransomware og efter et kort stykke tid, standse den aktive ransomware i 77% af tilfældene.

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring an M.Sc. in Engineering.

The thesis deals with ransomware, detection methods of ransomware, methods of mitigation and testing of live ransomware on virtual machines.

Lyngby, June-2017

Niels Beuschau

Jesper B. S. Christensen
Niels Beuschau

Acknowledgements

We would like to thank our supervisor Christian Damsgaard Jensen for the help, guidance and counseling throughout this thesis. Also a special thanks to virusshare for letting us download almost 40.000 different encryption ransomware for testing purposes. We are also grateful to Henriette Steenhoff who has lended a hand in the analysis of data. Furthermore we are thankful for the assistance that Nicklas Johansen provided in the development of the Game Theory sections.

And lastly we would like to thank Amirhossein Shahineelanjaghi and Morten Von Seelen, in their assitance in gathering information about ransomwares and live samples.

Contents

Chapter 1

Introduction

In the beginning, the purpose of viruses and hacking in general was either to show off your abilities or a proof of concept, to see if the hack would actually work [Hig97]. This developed into larger amounts of destruction with no gain for the attacker except the thrill and fame for doing so [Hyp11]. Attackers then started to create bot networks, where infected systems would become part of the bot network, which could be used for generating spam emails, Distributed-Denial-of-Service attacks and more, usually for economic gain [Hyp11].

People with malicious intentions have been exploiting people for hundreds of years, and the technological development has only made it easier. When these people realized the potential of exploiting people for money online, things started to develop much faster [Hyp11]. In the beginning of online exploiting, many fake anti viruses and anti spyware programs started to show up, they claimed to have found spyware and malware on the system, even though there were no malicious files, the discovery of these files were free, but it required a payment in order to remove it. Ironically the anti viruses and anti malware programs were the malware themselves [Mav+].

The success of the fake anti viruses lead to another type of malware that required payments, but this method had a much more aggressive approach than the previous malwares in order to secure payment. This malware, called ransomware, has two different types. The first type, called locker ransomware, locks the user from the system, preventing the user from accessing anything but the locked screen, this locker then demanded payment in form of either vouchers, purchases on specific sites or in some cases bitcoin payments [SCL15]. The other method, called crypto ransomware, starts to encrypt important files, such as word documents, business spreadsheets, vacation pictures and the likes. When the ransomware deems itself done with the encryption a ransom message is shown to the user, demanding a payment for it to restore the files. Along with the demand of payment is usually a timer that indicates a deadline for payment. If this deadline is exceeded the ransomware will either delete the decryption key, such that the files cannot be recovered, unless the encryption is cracked, or delete all of the encrypted files [Sga+16]. The timer in the ransom note and the pressure of the loss of files is a part of the tactic to make the victim pay usually seen in scareware, a method explained in greater detail in this thesis. The distributors of ransomware has no interest in the victims that does not pay, and gain nothing from victims that does not pay. They do not care about the encrypted files or anything upon the system, what they are interested in is the payment and nothing else [AGM15].

The aim of this paper is to create and analyze various methods that can detect when a crypto ransomware starts to encrypt the files on a system. The effectiveness of these methods will be tested to find the best detection method to detect a crypto ransomware attack. Instead of having a blacklist of signatures to prevent the ransomware from getting into the system, these methods will detect the attack as it begins, thus having an effective behaviour based detection method. The tool created will test different detection methods with focus on several parameters. The most important parameters are reaction speed, effectiveness and number of false positives. Reaction speed is how fast the detection tool detects that the system is being attacked, meaning that encryption of the files has begun, this is measured in how many files are encrypted before the tool reacts. Effectiveness is about how many different kinds of ransomware are caught by the detection method. False positives is the ability of the detection tool to determine whether the threat is real or if it comes from a regular program on the system. The goal is to create a tool that can detect crypto ransomware when it attacks the system and afterwards stopping and mitigating the attack. This tool will not detect a dormant ransomware that does nothing, nor will it detect when the system is infected with the ransomware, it will only detect when the ransomware is attacking the system.

Ransomware has been seen on everything from smartphones, smartwatches and electronic billboards to healthcare facilities. Most operating systems such as Windows and Unix based systems (Ubuntu, Debian, MacOSX etc) are all affected. This project is focusing on ransomware that is targeting windows. This has been chosen since windows is the most targeted operating system for ransomwares and also the most common operating system [Dat16].

This thesis is divided into eight chapters:

Chapter 2 The basic properties of a crypto ransomware is presented, this includes the industry and economy of ransomware, encryption methods and how the ransomware communicates with a given controller. Following this is some case examples of known crypto ransomwares.

Chapter 3 A thorough presentation and analysis of several known and documented ransomware detection, mitigation and remediation methods along with relevant theory.

Chapter 4 Here the methods for detection that have been considered implemented are described, this includes how they detect probable threats, possible flaws and potential methods of avoiding detection.

Chapter 5 Proposed methods for mitigation are described.

Chapter 6 This chapter describes the testing environment, the implementations necessary, test cases and the process of creating these.

Chapter 7 In this chapter the results are analyzed, and the effectiveness of the detection methods are measured. Furthermore a discussion that suggest how to optimize the detection method is made. This also includes a game theory analysis of interactions with ransomware.

Chapter 8 A conclusion for the thesis and the work that has been done during this process is made.

Chapter 9 Perspective for future works, not only for this project but ransomware detection in general.

Chapter 2

Primer: Crypto Ransomware

In this chapter the properties of crypto ransomware will be explained. First, a brief explanation of what crypto ransomware is, what it does and how big an industry ransomware actually is. Next, the methods of infection used by ransomware to become distributed as widely as possible is explained. Following this is an overview of the encryption schemes used and how the ransomware communicates with its command and control servers.

Crypto Ransomware is a type of malware that once it has infected a system encrypts user files. Then it demands some form of payment to decrypt the encrypted files within a given time limit. This payment is nowadays usually in bitcoins [TCM], where earlier it was in online shopping, premium telephone numbers or other payments difficult to trace [Win]. The costs for attacks that hit individuals are usually around 300$ worth of bitcoins, but for larger companies or institutions the costs can be higher, especially the cost of having downtime or the recovery can be quite expensive. As an example when The San Francisco metro was hit late November 2016 and was affected for a weekend the estimated lost ticket revenue amounted to 50.000$ [SFG], on top of that are the expenses for recovery and consultants. There are no limits to who gets infected by ransomware and the consequences varies a lot. The service sector is the sector among organizations most commonly infected, but almost every sector has been hit with ransomware on some scale, this includes hospitals, public transportation and police departments [16]. Crypto ransomware is a growing industry with a large number of infections each year as seen in figure 2.1, this leads to a large income for the distributors of this ransomware. As an example, the WannaCry ransomware affected large parts of the British National Health Services in beginning of May 2017, resulting in cancellations of scheduled surgeries and appointments [Bra].

In 2015, it was estimated that criminals earned around $24 million from ransomware from the United States, and in the first three months of 2016 $209 million in ransomware demands had been paid in the United States alone [Fin]. It has been said that for the ransomware CryptoLocker roughly 41% of the victims pay for the decryption of their files [Sco14], while the general payment percentage in 2016 was around 34% according to Norton [ONe].

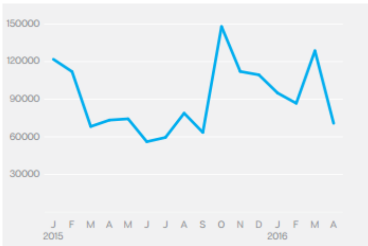

Figure 2.1: Overall Ransomware Infections by Month from January 2015 to April 2016 [16]

Although it is known what regions most of these ransomwares originate from [Hyp11], there are usually no specific targets for common ransomwares. Ransomware is distributed through various means. The most common ways are infiltration through email, web exploits by using exploit kits such as Angler, Drive-by-downloads, or extensive phishing campaigns [Ost].

The latest ransomware WannaCry also known as WannaCrypt, WanaCryt0r 2.0, Wanna Decryptor is a ransomware that hit the world the 12th may 2017, and is the first of its kind to utilize worm like behaviour successfully. It exploits a vulnerability in Windows computers with the Server Message Block (SMB) where it not only spreads to other computers online, but also spreads to other computers using the Local Area Network.

Most of the victims of ransomware are home users, this is largely due to home users not having proper security or backups, and therefore easily gets infected and has no other options than paying the ransom [IBM]. However the healthcare industry has been targeted by spear phishing campaigns [16], and latest was the WannaCry which hit the British National Health Services primarily due to old systems running windows XP.

Everything containing data can be hit by crypto ransomware, and everything with an interface can be targeted by locker ransomware. Ransomware that targets smartwatches and smartphones is usually locker ransomware and also a growing industry [MNS16].

For crypto ransomware, once the system is infected, the ransomware will start to encrypt files with little communication with the command and control server, if it still exists. This communication is usually performed over anonymity networks such as TOR or I2P, but can also take place using more normal connections such as HTTP or HTTPS [SCL15]. Some command and control servers are taken down such that the ransomware has nothing to post to, and how the ransomware reacts upon missing a command and control server varies. Some does not encrypt the files, because there is nowhere to post the encryption key, while some encrypt the files and tries to send the encryption key to a non existing server. The latter means that even if the ransomware payment is met, the files will remain encrypted due to lack of a decryption key.

The first ransomware, PC Cyborg from 1989, used a symmetric encryption to encrypt the files on the drive of the computer [Kas]. This was easy to decrypt since the encryption key was stored along with the encrypted files. In fact several ransomwares have been found to have a default encryption key for all files and all victims [Hay]. However, when looking at the newer generations of ransomware, such as Jigsaw, TeslaCrypt, CryptoWall, WannaCry etc., they usually use a combination of asymmetric and symmetric encryption algorithms. Normally a 256 bit AES symmetric key is used for encrypting files, and then an 2048 RSA asymmetric key is used to encrypt the AES key [Edi]. Using such an encryption scheme makes it theoretically impossible to decrypt the files without the decryption keys. WannaCry, which is the most noticeable ransomware in recent times, generates its encryption keys in the following manner. Once the system has been infected it generates an RSA keypair, where the private key is encrypted using a hardcoded public key from WannaCry and sent to the command and control servers. The public key from the newly generated keypair is then used to encrypt 128-bit symmetric keys used for each individual file. [Sym]

The method of encryption can be put into three different categories:

Category 1 This type of ransomware opens a file, reads the contents and then writes the encryption into the file, thus overwriting it. This means that the content of the file is encrypted, but not necessarily the file itself, the file might not even be renamed.

Category 2 The file to be encrypted is moved to another directory where the ransomware encrypts the file, then moves the same file back into the original directory. Here the file might also be renamed.

Category 3 Here the original file is read and a new encrypted file is made based on the original, next the original is overwritten or deleted [Sca+16].

After all of the relevant files have been encrypted a ransom note is delivered onto the infected system. This is sometimes done with an opened window that cannot be closed, another method of delivering the ransom note is changing the desktop background to the ransom note itself. The ransom note usually demands around $300, but this can vary from country to country [Sym15]. Most ransom notes explain what has happened and why the files are impossible to recover without payment. Usually there is a timer and other psychological effects to frighten the victim into paying, as seen in figure 2.2.

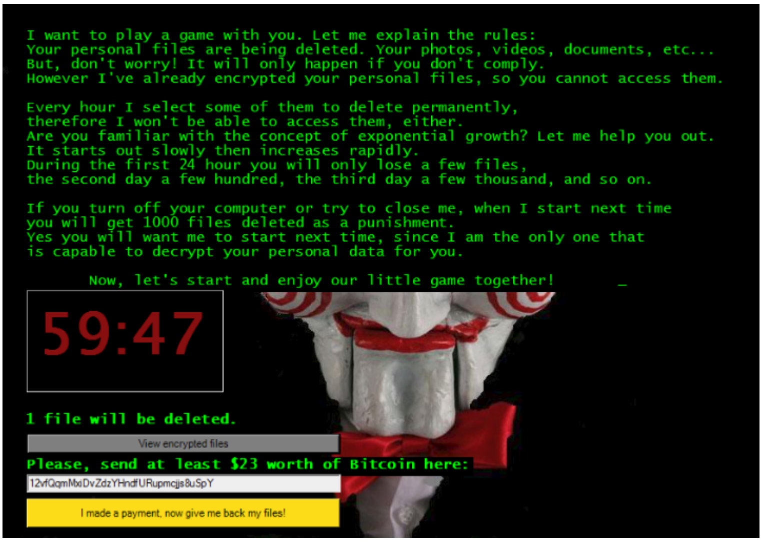

Figure 2.2: Jigsaw ransomware note

Since bitcoins and how to obtain them is not something commonly known, some ransomwares show guides and homepages of how to purchase bitcoins in order to pay the ransom as seen in figure 2.3. Some ransomwares even provide support and service hotlines.

Figure 2.3: Cryptowall bitcoin guide

2.1 Ransomware examples

CryptoWall is, as the name implies, a crypto ransomware that showed up in the beginning of 2014. It uses an AES encryption and then encrypts the key to the AES encryption with the public key of RSA keypair generated uniquely for every attack. Cryptowall is deployed through usual attack vectors, exploit kits, drive-by-downloads, phishing campaigns and email spam.

In order to ensure persistence, a ransomware, among other things, adds files to several different directories in the system that can start up the ransomware once more. These folders are usually folders not normally used by users such as the directories *appdata* and *temp*.

As seen in figure 2.4, the ransom note explains how the files have been encrypted and links to it such that the victim them selves can read about the encryption and why it is impossible to recover the files. Furthermore, the ransom note explains to the victim what has happened and why the only way to recover the files is by following the instructions. The ransom note even explains how and where to acquire bitcoins, as seen in figure 2.3.

Many locker ransomware uses psychological effects to frighten victims into pay-

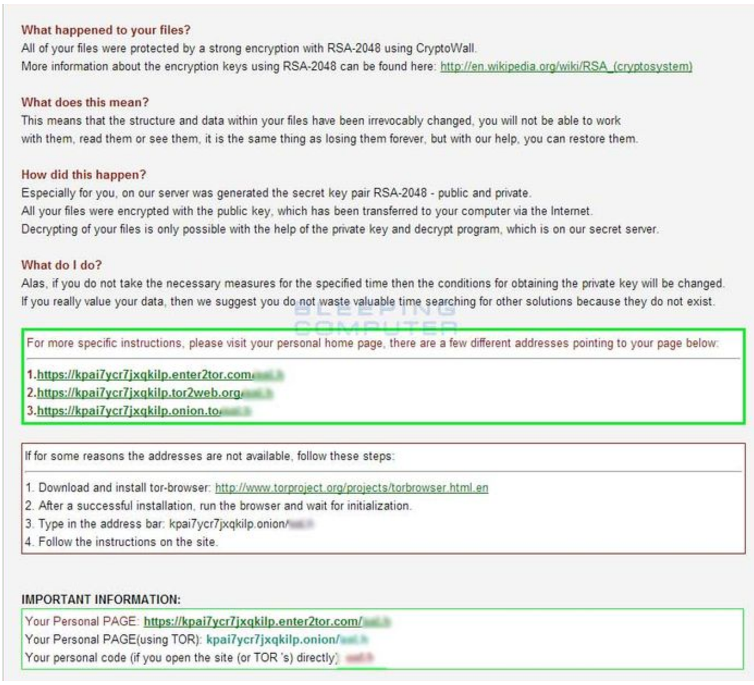

Figure 2.4: CryptoWall ransom note

ing to have their systems restored to normal quickly, they usually pretend that the locking of the computer is made by some law enforcement agency such as the FBI. They inform the victim that they have been caught performing an illegal action. Often the alleged illegal activity is downloading pirated movies, accessing pornography, or even child pornography. The locker ransomware informs the victim that the illegal offense could result in prison sentence or a very expensive fine. However, they offer a "first time offenders fine", which is a lot lower than the normal fine. This tactic scares the victim from seeking help from others, while also believing they are getting a "good deal". [Gam]

Crypto ransomwares do usually not rely on using fake governmental warnings, but they still use psychology to frighten the victims. In the jigsaw ransomnote, in figure 2.2, a timer is clearly shown, and if payment has not been received files will be deleted. This timer is meant to instill panic and urgency in the victim, increasing the probability for them to pay, since they do not have have time to research alternative options.

Another psychological feature, is the "show of good faith". Some crypto ransomwares offers to decrypt a few for the victim for free, in order to show that they are able to decrypt the files. This is supposed to make the victim trust the ransomware, and again, increase the likelihood of receiving payments.

Where some crypto ransomwares decrypt a file for the victim as a show of good faith, others use more threatening methods in order to make the victim cooperate. As seen in the jigsaw ransomnote it warns the victim not to shut off the computer or close the ransomnote, otherwise there will be consequences, usually deletion of already encrypted files.

It is important for an effective antivirus to know how a ransomware works, what it does and what kind of communication it makes with a server. To test what a ransomware does it is often simulated in a virtual environment or put into a sandboxing tool, from there every single action the ransomware does, can be monitored and analyzed. In order to prevent antivirus and other detection systems to test a ransomware in such a simulated environment some ransomwares feature anti-simulation techniques. How the ransomware detects it is in a simulated environment varies, but a know case is where WannaCry made a call to an outside domain that did not exist, if the environment returned with an answer then the ransomware would do nothing at all [End]. Other ransomwares have been known to act different on purpose in the simulated environment in order to throw off the detection method. In this thesis the ransomwares are tested on a virtual machine, by doing so the reaction and file encryptions can be monitored upon the machine. If a ransomware has an anti simulation method, either by not encrypting anything or somehow throw off the readings they might not be included among the ransomwares that the detection methods are tested upon.

2.2 Summary

To summarize, ransomware is a branch of malicious software that takes files as hostage and demands ransomware to release them. It targets individuals, corporations, organizations and public services such as hospitals and police stations. It is a growing industry which in 2014-2015, affected 131,111 users and 718,536 users in 2015-2016 according to Kaspersky Lab [Lab]. In 2015 ransomwares payments totalled 24 million $, and in the first quarter of 2016 it had increased to 209 million $, with an estimated total for 2016 to be 1 billion $ in the US [Dat16]. Some estimates show that the cost of downtime in the US in 2016 due to ransomware, cost upwards of 75 billion $ [Dat16]. In figure 2.5 is a timelime showing the enormous growth of ransomware families from 2011-2016.

The more advanced versions of ransomware contains anti-analysis techniques.

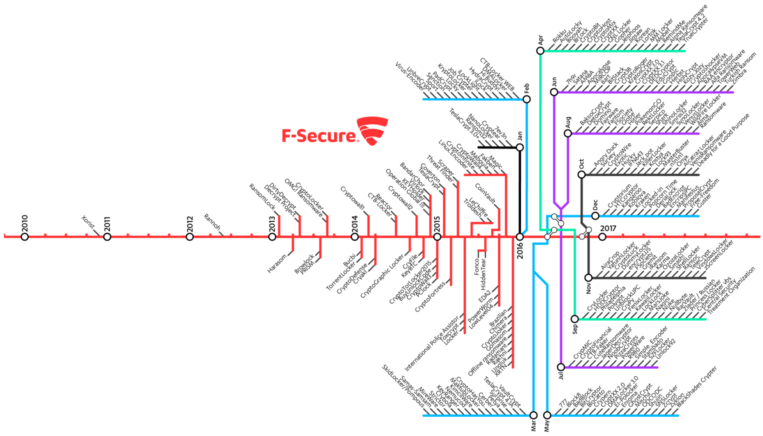

Figure 2.5: Ransomware timeline

This is because as with all software, ransomware also contains errors, which renders them less effective, by employing anti-analysis techniques these unintentional flaws are more difficult for security researchers to find. Examples of bugs is the usage of weak encryption scheme, not removing decryption keys from memory, or as recently seen with WannaCry, an unintended killswitch.

Chapter 3

Theory and related work

Through the literature analysis and analyzing the detection methods of current anti-ransomware products, several different methods for detection, mitigation and remediation was identified. This chapter presents others work and their findings divided into each of the methods.

3.1 Detection

3.1.1 Monitoring of File System Activity (SSDT)

It is possible to detect a ransomware attack by monitoring the file system activity as proposed and tested by A. Kharraz et al. [AGM15]. The proposed method hooks into the System Service Descriptor table (SSDT) and filters out interesting I/O request and their attributes such as process name, process id etc [AGM15]. By doing so, if a cluster of suspicious request are made, it is highly likely that the responsible processes are malicious. Furthermore, if a log of the SSDT calls is made it is also possible to remove everything the virus or ransomware has spread out on the computer. This can be done by finding a processes parents, thus finding the root of the problem and every single process or file these processes have made. Thereafter all of these processes are shut down and all the files removed, thus completely removing the ransomware code.

SSDT is an internal dispatch table in Windows, the table is used for system calls by the operating system. The information returned by the original operating system can be read or changed by hooking into the SSDT, a tecnique often used by rootkits and antivirus software.

The authors hooked into the I/O manager in the kernel and developed their own minifilter to filter read, write and attribute change requests [AGM15]. By utilizing the SSDT, the monitor is on level with rootkits and antivirus software, which leads them to argue that it will be very difficult for future ransomwares to bypass the monitor. Kharraz concludes that by analyzing and intercepting the I/O request they can reliably detect and stop a ransomware attack.

Not only will it be hard for future ransomwares to bypass the monitor, by having a system that hooks into the SSDT it is also very hard to remove since any I/O request is made to remove the monitor can be discarded by the monitor itself. Thus making it very hard to remove or shut down. This gives the detection method a very robust foundation.

3.1.2 Event Tracing Windows (ETW)

A research team from CyberPoint lead by Ben Lelonek and Nate Rogers held a talk at Ruxcon in 2016 and presented work on ransomware detection using Event Tracing for Windows (ETW) [Rog16]. Their approach was to analyze the events generated for file reads, writes and change in file size, and through an algorithm they developed a method for detecting ransomware. The algorithm is designed based on research they performed on ransomware behaviour, where they tried to find ways to generalize the behaviour of the variants. This generalization had a high number of false-postives, and was very dependant on Operating System delays, iterations etc. When looking at changes to the file size they compared original size vs. the encrypted size, this however also varied a lot due to different encryption algorithms, initialization vectors, and resulted in lots of false positives from benign processes. The behaviour when changing names, was rather consistent since most encrypted files would keep some form of their original name. The algorithm they developed was based on the explained research and works like this:

SuspiciousEvent=0;
if *File previously read* $\wedge$ *File just written* **then**
  if *Same PID* $\wedge$ *Threshold* $<$ *80 ms* $\wedge$ *File size delta threshold* $>=$ *1024 bytes* **then**
    SuspiciousEvent = SuspiciousEvent + 1
  end
end
if *SuspiciousEvent* $>=$ *3* **then**
  Filter false positives
  if *!false positive* **then**
    Handle process
  end
end

According to their tests, they are able to detect every ransomware. However, the solution has some limitations. At least three files needs to be encrypted before the system detects and stops a ransomware. Because the system is based on dynamic capture of events the performance can vary greatly and is subject to minor delays. Lastly, the authors also mention that it is not hard for future ransomwares to detect this type of monitor, since windows keeps track of all event listeners and therefore a ransomware could just check for any processes monitoring the logs.

3.1.3 Honeypots

The use of honeypots to detect malicious system activity was first proposed by [Bow+] and [Yui+04], and later implemented against ransomware in [Moo16].

Chris Moore has been using monitored honeypots to detect malicious system activity [Moo16]. The way honeypots work is by having files placed onto the system, that no program nor user would ever tamper with. The first honeypot ideas were more traps and bait than anything else. The intention of these were to be decoys and confuse an intruder, and when the intruder accessed the honeypot file a system would react and know that an intruder was in the given file. This can also be implemented to detect ransomware, this method would use the honeypot as bait. Since a ransomware is encrypting all files in every relevant folder it would naturally also encrypt the honeypot files, thus alerting the system that a program is tampering with the honeypot. A program called EventSentry can be used to make real time event log monitoring and monitor Windows Security logs. This can be used to raise flags when the number of suspicious actions reaches a certain threshold. A folder, made entirely of honeypots is created and monitored by EventSentry in order to capture unauthorized attempts to access objects in the folder. By using a single folder this also ensures some protection against false positives, as the user knows what folder not to tamper with, hence the only object that would tamper with that given folder is malicious programs. Along with this monitor is a tiered response to detection such that different amounts of attempts to access the honeypot files leads to different reactions. The more attempts detected the more severe the reactions, starting with sending an email to the administrator that there has been changes in the monitored folder, to determining and disabling the user or station that is hosting the attacking ransomware. Then disabling the network services, ending in shutting down the server, in order to protect the server from additional encryption by the ransomware. The tiered response is implemented in order to ensure minimum trouble for a user if the user would trigger the honeypots, but at the same time prevent further spread of a possible ransomware.

3.1.4 Machine learning

Diane Duros Hosfelt has made a machine learning method to detect when cryptographic algorithms are compiled [Hos15]. Algorithms such as SHA1, DES, MD5, AES etc. This detection method can be used to detect when crypto ransomware attacks the system and starts encrypting files. Diane Duros Hosfelt uses the Intel's Pin dynamic binary instrumentation (DBI) framework to identify and extract features. This injects code into the executed program in order to analyze the behavior of the program at runtime. If this code injection is detected by the malware it can avoid running the code thus avoiding detection. The machine learning method has only examined C and C++ code, but this problem is easy solved since the the model can be trained to detect and classify other language binaries.

Kharraz et al. [AGM15] analyzed a lot of ransomware families and how they interacted with a Windows system. They proposed monitoring Windows API calls such as encryption libraries, defragmentation API and more. The problem with this however, is that a lot of benign software uses these as well and could therefore create too many false positives. To combat this, the authors suggest training a classifier and thereby learning how to distinguish between benign programs and malicious ransomware. Furthermore, Kharraz also proposed looking at changes to the Master File Table (MFT), which keeps tracks of all files on the system. Through their analysis they conclude that it might also be possible to use Machine Learning to identify malicious changes to the MFT.

3.1.5 Monitoring of shared fundamental behaviour

Several other researches have analyzed some of the fundamental behaviour ransomware exhibits. This is behaviour related to deleting backups, ensuring persistence, and use of microsoft cryptographic API.

Monika et al. found a set of common registry keys that are either read or modified [MZL16]:

```
HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

HKLM\Software\Microsoft\Cryptography\Defaults\Provider Types\Type 001
```

The first is usually modified for programs to ensure they are started at boot, while the last one is read to access window's cryptographic API.

Similarly Ahmadian et al. found 20 common features among the most widespread ransomwares families [AS16]. These features cover folder access, registry changes and process calls. Ahmadian was able to, rather reliably, detect new ransomwares based on the 20 features. They do however, note that ransomware would be able to change their common behaviour, which would render most of the identified features useless. They do argue though, that any successful ransomware will have to access and delete files from Windows volume shadow copy service (vssadmin), which they track and would be able to catch all ransomwares doing this. They assume that if the ransomwares does not interact with vssadmin, then the user should be able to recover their files using the service, however as described in section 3.4 this might not be the case.

3.1.6 Antivirus

One of the most common protections employed against malicious software is antivirus software. A lot of different companies develop and sell antivirus software which usually use a combination of heuristics- and signature-based detection. It normally works by having a database of extracted signatures of known threats. When a file is executed it goes through the on-access scanner where it is analyzed and its signature compared to the signature database. Furthermore its code gets analyzed in the heuristic module. This combination allows antivirus to fairly well identify known threats and some new. However, they are not very efficient against ransomware. The problem is, unlike a keylogger which hooks into the keyboard input or a backdoor which creates e.g. a reverse SSH tunnel, ransomware does not exhibit these types of behaviour. In most cases, it is just a normal program which is able to encrypt files and send traffic over the TCP/IP protocol.

3.1.7 CryptoDrop

Nolen Scaife et al, has created CryptoDrop that monitors real-time change in user data in order to detect ransomware attack [Sca+16]. CryptoDrop uses three individual ransomware attack indicators in order to reduce the number of false positives and at the same time tries to keep the number of files encrypted by the ransomware to a minimum.

Filetype: Files rarely change their file type or formatting except for when they are encrypted, thus by monitoring changes in file types could indicate an attack, although a single change in a file type is not enough evidence to indicate that an attack is happening, therefore it takes several of these changes before a flag is raised. Adjusting these detection thresholds to the optimal solution takes a lot of testing on multiple different ransomwares.

Similarity hash: Since encrypted files are nothing alike the original files the content of these files can be compared with some similarity measure. By using similarity-preserving hash functions one can look at how different a file is before and after being written to [Kor]. If the similarity hash is highly dissimilar in many files within a specific timespan then a flag should be raised.

Shannon entropy: The assumed value of information in a message is called Shannon entropy. Since encrypted data always have a high entropy, this means that if many files have a high Shannon entropy as a result of being changed, then this could indicate that a ransomware attack is in progress. Shannon entropy will be explained more in detail in section 4.6

These three methods are the main methods CryptoDrop uses to detect ransomware attacks since most ransomwares triggers all three of the main methods. Furthermore CryptoDrop also raises a flag if there is deletion of several files since this could also indicate malicious activity.

The advantage of combining these individual detection methods is that if one is to be avoided it would trigger the other indicators much easier. This means that if future ransomwares are to avoid all three detection methods it requires a lot of time and some very good engineering in order to evade all the detection methods [Sca+16].

3.2 Mitigation

In the previous section, we covered how to detect an ongoing ransomware attack, however, once detected the attack should be mitigated. There is very little academic research on how to mitigate ransomware, since it is usually straight forward. The two primary ways of stopping a ransomware attack is either suspending or killing the malicious process.

Suspending processes can work well as you can temporarily stop a process believed to be malicious and either do further analysis, automated or manual. Furthermore you can ask for the user to decide, ensuring the process can not do any harm until the users takes action. The disadvantage of this is relying on the user acting correctly and with the right knowledge of which files might potentially be malicious on his/her system. It might just become another pop-up box indoctrinating users to always click yes or no, without much thought. This could either allow the ransomware to run rampant or shutdown falsely identified malicious processes. The problem increases if all the processes spawned by the ransomware gets suspended, which could lead to a dialog box spam. An example of suspending processes is the free tool RansomFree developed by CyberReason. It suspends a process identified as malicious, and requests an action from the user, to either allow or stop the process.

The advantage of directly killing processes, is that the malicious process is stopped right away, without interfering with the users normal actions or work-flow. However, the margin for error is significantly lower since stopping a non-malicious process could result in loss of work or system instability.

For both mechanism, all processes related to the malicious process should be handled at the same time. If not, some ransomwares might perform a revenge action, such as jigsaw deleting up 1000 files upon reboot [Mic]. This means, that all information about processes should constantly be logged, e.g. what other processes are spawned or what files are created etc. Having that information would allow the mitigation software to correctly stop the ransomware attack without any counteractions from the ransomware. An example of such a mitigation method is the one used by SentinelOne's EndPoint Security. They collect and track what all processes performs of actions, and once one of them is detected as malicious they take appropriate action against all processes created by the malicious one including its own parent and the parents' children.

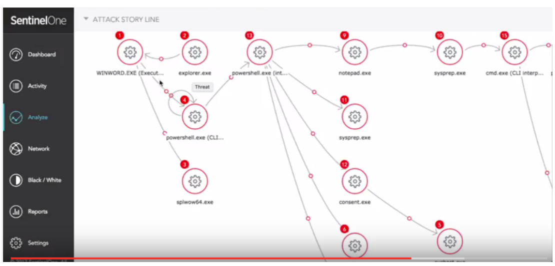

Figure 3.1: SentinelOne process tree example.

When the process has been stopped, cleaning of any persistence and other changes should be performed is covered in section 3.3.

3.3 Remediation

Remediation covers not only removing any forms of persistence, but also removing the files that were added to the system, which, by accident, could start the attack again. It also includes undoing changes to the registry database and attempted recovery of lost files. This section will cover how some of the commercial proprietary products work to remediate a ransomware threat.

There are several commercial products working as full protection suites, so they encompass detection, mitigation and remediation. Since they are proprietary products, not much besides what the companies say about their products is known, and no scientific articles has been released on their effectiveness. Nonetheless, this section will cover how some of such systems work, based on the information available.

SentinelOne uses a multi-layered approach which, as they call it *"covers the entire threat lifecycle"* [Sen]. Their approach is not based on signatures or heuristic analysis, but on a dynamic analysis of processes' behaviour. This dynamical analysis is supported by proprietary algorithms and machine learning, what is known though, is that they look at calls to the Windows volume shadow copy service (vss service), and blocks those that are not by their product or signed by Microsoft. They continuously monitor all processes and log their actions, and when one is deemed malicious, they kill the process and all of those related to it, such as its children, parent and parents' children. Since all the actions of the process are logged, they can easily revert the changes, which only leaves the files the ransomwares manages to encrypt. These are restored using the vss service which is able to recover them from the last time a snapshot was taken.

Checkpoint also have a commercial product by the name SandBlast Anti-Ransomware [Che], which for the most part works very similar to SentinelOne. Without knowing their proprietary algorithms, the primary difference is instead of using the vss service, Checkpoint uses their implementation of a service similar to vss.

3.3.1 Decryption tools

Most ransomwares uses strong encryption such as AES256 and RSA with a 2048 bit key, and known cryptographic libraries such as the one in Windows or open source options.

A few poorly constructed ransomwares do not however, and usually security researchers are able to find flaws in their own developed encryption schemes allowing the files to be decrypted. In other cases, some ransomwares, even though they use strong encryption, have the key stored within the ransomware, again allowing security researchers to find it.

In more recent cases, with e.g. WannaCry, researchers found a way extract the encryption key from memory because it was not properly removed from memory, so as long the computer was not shut down, the key could be extracted. Another set of researchers from Kaspersky Labs [Lab] found that poor coding skills could allow recovery of files lost to the encryption due to how WannaCry deletes files.

All of these flaws, allows security researchers to develop decryption tools which are released to the public for free. Nomoreransom is a collaboration between National High Tech Crime Unit of the Netherlands' police, Europol's European Cybercrime Centre, Kaspersky Lab and Intel Security and works by collecting all the developed tools in one place to help ransomware victims.

3.4 Windows Volume Shadow Copy Service

Windows Volume Shadow Copy Service, also known as VSS or VSC, is a system for creating snapshots of disk volumes. It works at the disk block level, and works by tracking all changes to the blocks. If a change on a block is about to happen the block is backed up before the change. Seeing as it is used as snapshots of the volumes, the VSC only ensures that blocks, and files therein, can be reverted back to when the snapshot was taken. This means that if a file is changed several times after the snapshot was taken, the newer changes are not recoverable, unless a new snapshot is taken in between each file change. One of the advantages of working at the block level, is that if a file is deleted, the VSC does not need to create a copy, only if the blocks it resides on is about to be overwritten [Szy].

The VSS has a limited amount of disk space to store the snapshots in, usually 5% of the main disk. There is no limit to the amount of snapshots that can be taken, as long as the total size does not exceed the limit. If the service tries to create a new snapshot when there is a lack of space, then starting from the oldest, the snapshots gets deleted until there is sufficient space for the new snapshot. In the case that there is not enough space for the latest snapshot, all snapshots are deleted, since the VSS does not store partial snapshots [Szy].

Previously it has been explained that some detection and remediation methods rely heavily on VSS. The concept is that, if a ransomware want to be truly effective, it has to clean/disable the VSS storage. In order to this, it has to perform API calls to the VSS which can be monitored and manipulated, resulting in the ransomware being detected by those actions.

At first, it seems like a perfect approach to always monitor calls to the VSS and act accordingly, however, the VSS methods contains two problems. The first being, that if it is a long time since the last time a snapshot was taken, recovering files from the snapshot could still involve a lot of lost work. The other is a theoretical attack on exhausting the VSS disk space. A future ransomware could instead of calling the VSS API, instead delete enough files and overwrite their blocks on disk with random data. This would force the newest snapshot to grow in size, and at some point having to delete old snapshots. Continuing this attack, would end up forcing the snapshot to delete itself since no partial snapshots are stored. Obviously the ransomware should not delete normal documents and spreadsheet which is of value, but rather large programs.

3.5 Game Theory

Game theory is about any interaction between multiple entities often called players, in which each entity's payoff is affected by the decisions made by others. It is used in a wide range of fields such as economy, politics, biology, military, psychology and computer science. Detailed below are several concepts used within game theory to describe the interactions which are relevant in the context of ransomwares. Some of the concepts requires more explanation and as such also have their own section going further in-depth.

Static game is where each player chooses their strategy simultaneously from their respective strategy space. The combinations of these strategies then determines each players payoff. Even though the strategies are chosen at the same time, does not mean that they are executed simultaneously.

Normal form games is way to describe a game where you know all the players, their strategies and their payoffs.

Complete information means that players payoff functions are common knowledge. That is, for each strategy that player I could play, player J knows the payoff. And player I knows, that player J knows his payoff. And player J knows that player I knows that player J knows his payoff, and so on.

Strictly dominated strategy is when a strategy **s'** is strictly dominated by another strategy **s"** if for each feasible combination of other players strategies, the payoff from playing **s'** is less than that of playing **s"**.

Iterative elimination of strictly dominated strategy is a method for analyzing games, it works by eliminating strictly dominated strategies. It is often used to reduce the complexity of games, and number of calculations. Sometimes it can even solve the game.

Nash Equilibrium is one of the central analysis methods within game theory. It is known for being one of the best methods for predicting game outcomes. See section 3.5.1 for a more in-depth explanation.

Pure strategy and Mixed strategy: A mixed strategy is the probability distribution over all of the strategies of that player, usually in the form (q,1-q), where 0<=q<=1. In case of q being 0 or 1, then it is a pure strategy.

Best response is the best strategy a given player can play which produces the best expected payoff, taking the other players strategies into account.

Expected payoff is the value a given player is expected to receive by playing a given strategy. Expected payoff is calculated by multiplying the probability with the payoff.

Dynamic games is where the players choose their strategy in turns and the actions are executed in sequence. I.e. company 1 chooses to produce quantity q1 and then company 2 observes q1 and chooses their quantity q2.

Repeated games is usually where a fixed group of players plays a given game repeatedly. The outcomes of all previous games is observed before the next play begins. The idea is that credible threats and promises about future behaviour and strategies, can influence the current behaviour.

Perfect information is games where each player at each move knows the full move history so far.

Imperfect information is where the full move history is not known.

Sub game as the name implies, this concept, is where a game unfolds within a game. Recall that in dynamic games players take actions in turns, in subgames players can take simultaneous actions.

Backwards induction is a method applied to dynamic games to analyze the outcome. When using this methods, the game is always solved from the the last action. All strategies with their payoff is put into a tree as shown in figure 3.2. The top payoff in the pair of payoffs at the end of each branch of the game tree is player 1's and the bottom is player 2's.

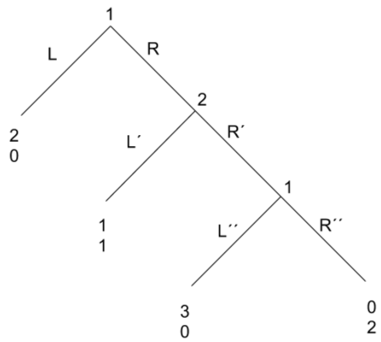

Figure 3.2: Extensive form tree usually used in backwards induction

Non-cooperative the players play against each other in a competitive way. Non-cooperative games are often analyzed by predicting the individuals players strategies and payoffs using methods such as Nash Equilibrium.

Cooperative means that the games can be considered as a game where the players have to work together such as in a coalition which is commonly known as cooperative.

Zero-sum are games where the the sum is 0. In zero-sum games if a strategy is beneficial to one player, then it is at an equal expense of another player, such as in poker games.

Non-zero-sum are games where the payoff gained by one player, is not at the expense of another player.

3.5.1 Nash Equilibrium

Nash equilibrium is a fundamental concept within game theory to analyze games. Assuming a static game with pure strategies, then a 2-player game is in Nash equilibrium if:

- Player 1 makes the best decision he can, taking into account Player 2's decision, while Player 2's decision remains unchanged.
- Player 2 makes the best decision he can, taking into account Player 1's decision, while Player 1's decision remains unchanged.

"Prisoner's Dilemma" is a well known example of a non-cooperative game. It shows, that even though the best outcome for both players is to stay mum and thereby minimize their prison sentence, then when analyzed with Nash Equilibrium, the best strategy to play is snitch, resulting in a higher prison sentence for both, which seems counter intuitive. Two prisoners, Alice and Bob, were arrested for committing a crime. Dependant on if they choose to be mum or snitch they have the following options:

- If either Alice or Bob snitches the other does not, they will be granted immunity, or 0 years in prison, however, the other will get 10 years in prison.
- If both Alice and Bob both snitches, they will both get 5 years in prison.
- If neither Alice nor Bob snitches, they will both get 2 years in prison.

They both know the options and are then split up so they will not know what the other will answer. They know the payoffs of the others strategies, and they choose simultaneous, so it can be considered as static complete information game. Their options can be put into the grid seen in figure 3.3.

Alice / Bob	Mum	Snitch
Mum	2/2	10/0
Snitch	0/10	5/5

Figure 3.3: Prisoners dilemma choice grid

The intuitively best option is if both of them stay mum, since the total prison time will only be 4 years, also know as the social optimum. However, if e.g.

Alice snitches, then Bob will get 10 years in prison and vice versa. According to Nash equilibrium, Alice should make the best decision she can, assuming Bob is taking the best decision he can. Since individually the best decision is to snitch, Alice should assume Bob is going to snitch. If Bob snitches and Alice does not, she will get 10 years while Bob gets 0 years, which means, she should also snitch, resulting in both getting 5 years.

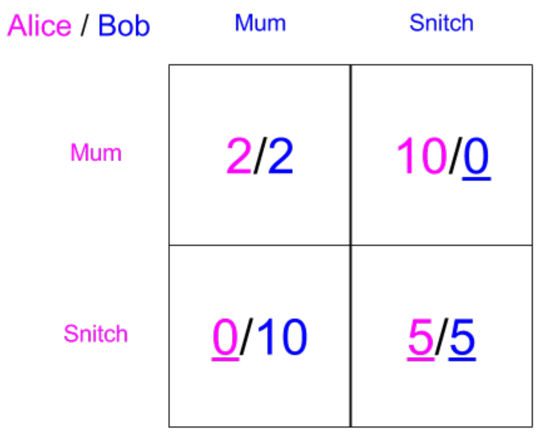

Figure 3.4: Prisoners dilemma choice grid with best-response underlined.

As shown in figure 3.4 the best-response, represented by the underlines can be seen. Here we see the Nash Equilibrium (snitch, snitch) which provides the payoffs (5,5).

The reason that this game is so popular to use when describing game theory is because they both end up with snitching on one another, resulting in 5 years each, which is counter intuitive compared to the 2 years they could have gotten by being mum. This is because snitch is a strictly dominated strategy i.e 0 years are better than 2 years in prison.

This is the overall idea of Nash Equilibrium, and it will be used to analyse the interactions with ransomware.

Chapter 4

Methods for detection

This chapter will discuss and analyze the different detection methods and how some of these have been implemented. First, the different methods will be presented, how they work and what they do. These methods will be analyzed theoretically to give an estimation upon the different qualities of the detection method, and how well they would detect a crypto ransomware, theoretically. This analysis is somewhat based upon information gained from related work. Furthermore, for each detection method it will be discussed how a ransomware can avoid detection and thus avoid triggering the detection method. Following the theoretical aspect is also an explanation of how the detection method has been implemented, if deemed achievable to implement within the capabilities and timelimit. Next is a discussion on how to avoid false positives with the given detection method.

4.1 Honeypots

4.1.1 Theoretical

A typical honeypot when talking computer security is a server set up to look like a legitimate regular server. But this server is often on its own network while being monitored. Upon the server is typically also some false information that takes an effort to acquire, thus luring the attacker to use exploit tools in order to obtain that information. All of this is monitored and saved such that an antivirus will know such an attack in the future.

This thesis will also make use of honeypots as a detection system, although the honeypots are files instead of a server. These files are placed among regular data, but monitored by a system that checks for changes made to these files. If the honeypots were placed to catch regular hackers that looked for credit card information or passwords, the honeypot files would be named *passwords.txt* or something similar to catch attention. Against a ransomware the contents of a file does not matter, since all the ransomware does to the files is encrypting them. Therefore the honeypot files in the directories are multiple files of different size and type. This is done to detect if a ransomware targets specific files or encrypts them in a unique order.

Using honeypots to capture ransomware just means they need to be there, eventually they will be encrypted by the malware and that is when the system monitoring the honeypots would react to a change in the honeypot. Naturally, the faster a honeypot is targeted by a ransomware, the faster a detection method would react and begin the mitigation process. If the honeypots are placed randomly, then the more honeypots there are, the faster a ransomware should be detected due to the higher probability of a ransomware encrypting a honeypot. From what has been observed so far, the ransomware does not pick the files to encrypt randomly, but what looks like alphabetically in most cases. By observing what files the ransomware encrypts and in which order, one can deduct a pattern that the ransomware follows. If a ransomware always encrypts in alphabetical order, it would be natural to place honeypot files at the beginning of every directory. Whereas if the ransomware encrypt the smallest files first, then the smallest files in a file system should be honeypots. This idea is explored further in section 7.3 which covers Game Theory.

4.1.2 Implementation

First, a system that is able to monitor changes in certain files in a directory was needed. For this, filemon was used. Filemon actively monitors files containing a given predefined string. The string chosen for all honeypots in all of the directories on the tested computers was chosen to be *honeypotbait*. As long as a file contains that string, no matter what type of file or what else their name is, filemon will monitor changes made to that file, whether it is deletion, change, creation or merely renaming. Filemon can be programmed to react upon multiple different changes in the file, change of size, name, attributes etc. The implemented filemon has been programmed to monitor the last write to the file, change in the filename and changes to the size of the file. The code for filemon can be found in appendix E.3.2

Once a change has been registered in a honeypot file, filemon registers it. A user who has installed a honeypot based ransomware detection system would refrain from changing the files, one can argue that there is a high probability that if a honeypot has been changed then it is not the user but something malicious. Despite this the implemented program has a threshold of two honeypot files being changed within a minute to react. This was chosen as a user may accidentally delete or somehow change a single honeypot file, but if several files has been changed within a minute then there is a high probability of a malicious attack, at least when dealing with a regular user. This also means that a ransomware that does not change two honeypots within a minute will not be detected by this method. One can argue that a ransomware that only encrypts a few files every minute is a very slow working ransomware, although still a ransomware. The way a ransomware encrypts files and how to detect this using honeypots is discussed in section 7.3 Once the threshold has been met, the filemon will react to this and start shutting down the process that has tampered with the honeypot file as described in section 5.1.

4.2 Monitor processes that tampers with vssadmin.exe

4.2.1 Theoretical

Ransomwares will in general try to delete any backups if possible, since this increases the incentive for the victim to pay. A sort of backup service exists on Windows, it is called Windows Volume Shadow Copy Service or VSS for short. It takes a snapshot of the disk from time to time, and allows files to be reverted back to the state they were in at the time of the snapshot, a more detailed explanation of VSS can be found in section 3.4.

Most ransomwares usually tries to delete all snapshots or disable the VSS, in order to ensure that the encrypted files cannot be recovered, and thereby increase the chance of a payout.

```
vssadmin.exe Delete Shadows /All /Quiet
```

The code snippet seen above shows how a simple call to the VSS can delete the entire *"backup"* provided by Windows. Since it is crucial for ransomwares to delete this backup, it should be possible to monitor I/O calls to its process, vssadmin.exe, in order to detect or prevent a ransomware from deleting the backup. By blocking such a call, not only would the recovering of encrypted files be possible, but the blockage of an unsigned process calling vssadmin.exe requesting for deletion of every snapshot is very suspicious and a clear indicator of malicious activity. This would currently work well for most ransomwares, however as discussed in section 3.4 this might not be the case for future ransomwares.

4.3 Monitor commonly targeted folders and registry

4.3.1 Theoretical

It seems most ransomwares target the same folders, since that is where the users data is, and the same registry keys since they contain references to Window's cryptographic API, start options and more. If a lot of ransomwares share the same behaviour it would make sense to monitor that type of behaviour. However, this method has two significant problems.

The first one being, that accessing common folders and creating/reading/deleting files from them, is very common behaviour and would most likely be prone to a lot of false-positives.

The second problem is that registry changes are often used for making the ransomware more lightweight and easier to develop. If anti-ransomware software started to monitor access to Window's cryptographic API then most ransomwares would probably just shift to some sort of open source implementation. Likewise, instead of ransomware gaining persistence using some default start options built into Windows, they could do it through various means such as injecting themselves into other programs. This would most likely raise the complexity for ransomwares and require more development time from their authors in the beginning, but it is not unlikely that ransomware frameworks would incorporate these features.

All in all, a detection method based solely on this, would either result in a lot of false-positives or a sort of cat and mouse game. This method is therefore very unlikely to be successful on its own.

4.4 SSDT calls

4.4.1 Theoretical

By hooking into the SSDT calls upon a system one can monitor almost every action there is upon a system. By having such a tool at hand the next step is to create algorithms that can recognize a ransomware attack, whether it is by detecting several encryption patterns or other indicators of a ransomware infection and attack.

These algorithms that should be able to recognize a ransomware attack needs to be fine tuned and needs to know exactly how a ransomware attack looks like in SSDT calls. Specifically the algorithm should be able to identify when an encryption is happening, since that is a requirement for a ransomware. How the encryption pattern is identified can be different for each encryption method. One could use machine learning and simulate several ransomware attacks in order to train a machine to recognize the attack when it starts.

It is however, not unrealistic, to argue that ransomware developers could develop new ways to encrypt the files, and thereby making the SSDT method obsolete against new types of ransomwares.

4.5 Monitor high resource consumption

4.5.1 Theoretical

The faster a ransomware wants to encrypt, the more resources it is likely to use. Usually it would have a high CPU and harddisk usage. The CPU usage would increase due to running the encryption algorithm scheme, and the harddisk usage would increase, since it both needs to read all the files from the drive, but also write the encrypted files to the disk.

It might therefore be plausible to detect ransomware based on this method. It is not unlikely that due to other detection methods, ransomwares in the future might try to read as many files as possible into the RAM to avoid detection while encrypting files, and then only once the RAM is used, would it write all changes to the disk right away. This allows monitoring of CPU, harddisk and RAM to be a theoretical possibility.

This method, might be prone to a lot of false-positives though, since installing a game or large software package such as Microsoft Office, might also use a lot of all 3 resources. So as a stand alone method, this probably would not work, however in a tiered solution, it might add to the credibility of the threat score.

4.6 Shannon Entropy

4.6.1 Theoretical

The entropy of a file is a measure of the distribution of bytes in that file. A byte can be any value from 0 to 255 depending on what the byte is representing. A normal text file would have many bytes representing the values of the alphabet, but not many bytes for special characters. This means that the bytes in a normal text file is in a disorder and not evenly distributed. Normal texts in most languages have letters that occur more often that others, for example *e, a, s*, etc. where special characters such as £$§ are uncommon in a normal text. A normal file has a high difference in the different bytes. When a file is encrypted the bytes are randomized and distributed very differently and probably very even. This can be measured and calculated in order to test whether a file contains an approximately even distribution of bytes or an imbalanced one. By measuring this for a file we would be able to give an estimation of whether the file is encrypted or not. The formula for calculating the entropy for a file is given in equation 4.1 where p_i is the probability for a given byte. The formula returns a value between 0 and 8. Where 8 means there is a perfectly even distribution of bytes over the file. Meaning the higher the entropy the higher probability of an encrypted file.

$$e = \sum_{n=0}^{255} p_i * log_2(p_i) \tag{4.1}$$

The probability for a given byte, p_i, is calculated by counting how many bytes of that type there is in the file, divided by the total number of bytes in the file.

In order to make the entropy a number between 0 and 1 the original entropy has been reduced such that it fits between 0 and 1 as seen in equation 4.2 and

4.3 .

$$e = \sum_{n=0}^{255} p_i * log_{256}(p_i) \tag{4.2}$$

$$log_{256}(x) = \frac{log_2(x)}{8} \tag{4.3}$$

The problem with the file entropy, is that for larger files the entropy is naturally high. Most books have an entropy value between 0.8 and 0.9. Compared to that most encrypted files have an entropy value above 0.98. Files three of four times larger than a regular book usually have an entropy above 0.95. This means that files of that size cannot be separated from encrypted files when comparing them on their entropy.

By looking at entropy of the files before and after a write action has been done to that file, we should be able to determine if that file has been encrypted. If a file's shannon entropy changes significantly, i.e. if an entropy value of 0.3 suddenly changes to 0.98 it should be a clear indicator of file encryption.

The shannon entropy has a potential faster detection time than the honeypots, since it tests every single file whenever there is a change to them. Where the honeypot detection method requires the honeypot to be targeted by ransomware. The problem with our version of the shannon entropy might be that for every file that has been changed, the program needs to read every byte in that file and then parse it into the correct entropy, this might cause a delay in speed, and if the file is locked, then it is not possible to read the bytes of that file.

4.6.2 Implementation

The first thing the shannon entropy detection method ought to do is finding the shannon entropy for all files in the directories and store these values. For the shannon entropy to know when files are tampered with, a monitor of created, changed, deleted and renamed files is needed. Since filemon is already installed for the honeypot files where it monitors honeypot files only, it has been modified to the shannon entropy where it monitors every single file. In order to avoid false positives and a detection method that reacts if a single suspicious action is made, a threshold has been implemented. This threshold varies from the different versions of the shannon entropy detection method, but is made such that every suspicious action is counted and will trigger a reaction once the threshold is met. If a file is newly created and it has a large entropy then it counts towards the threshold of the shannon entropy. Likewise if a file is changed and the changed file has a much higher entropy than the original, then that too counts towards the threshold. To figure out how much larger the entropy of a file must become in order to be suspicious a data analysis has been made. The entropy of every single file in the directories has been saved. A ransomware then encrypts every file in the directories and the entropy of those files are taken. The original entropies are separated into several different categories based on size, each category is then measured upon how much the average entropy has changed when the files have been encrypted. This determines how much a file is allowed to change without counting towards the threshold. The categories can be seen in appendix E.4.3.

When a file is created in the system, the shannon entropy searches a dictionary for a file of similar name in that dictionary, if such a file exists and the entropy is the same as the other file, then it must have been a copy action or a move action. That should not raise any suspicion. If a file is changed, the filemon informs about the change and what file, the program then takes the entropy of the changed file and measure whether it is suspicious or not. The shannon entropy does not react upon many files being deleted, although that is possible with the filemon implemented.

False positives is a high risk when using shannon entropy, since pdf's have a natural high entropy that might cause the detection method to react upon pdf files being created or changed many times within a short time limit. Since the shannon entropy looks at changes at every single file, it cannot be avoided by the user that the shannon entropy will test every file the user changes. This might result in a higher probability of false positives.

To avoid being detected by this method, a ransomware should either lock the encrypted files, such that the detection method cannot calculate the shannon entropy of the changed file. Otherwise the ransomware needs to encrypt a file, but still keep the change in the shannon entropy relatively low. This requires either a weaker encryption method, which can be broken easily, or a specific encryption method that keeps the change in the entropy low while safely encrypting all the files in a way such that they cannot be decrypted.

Chapter 5

Mitigation Techniques

In section 3.2 some of the advantages and disadvantages of either suspending or killing a process has been covered. The primary difference is the interaction with the user. The user is deemed not to be trusted to make the right call, and therefore the process will be killed as soon as it has been identified as malicious. It is not necessarily straight forward to identify what process is tampering with a file and thereby which process is the malicious one. Our proof-of-concept implementations for example, make use of third party program called Process Monitor or procmon for short, there are other methods though, such as using SSDT.

5.1 Procmon

The steps in a ransomware detection and mitigation tools is first to detect that there is a ransomware encryption occurring, then figure out what process is performing the encryption and lastly, terminate that process. The problem with these three steps is the middle part, to find out what process is encrypting the files. C# does not have a single tool for registering what process has changed a given file. Therefore, in order to identify the process responsible for encrypting the files on the computer, the answer is either to change programming language such that the mitigation tool can dig deeper into the layers of the computer or use a third party program that has the tools to monitor process activity.

Procmon is a monitoring tool that shows all desired activity within the system. Since events constantly occur, Procmon has the ability to enable filter such that the user does not get flooded with information when using the program. Such a filter could include or exclude processes with certain names, read/write operations on files and more. A sample of a set of filters we had is seen in figure 5.1.

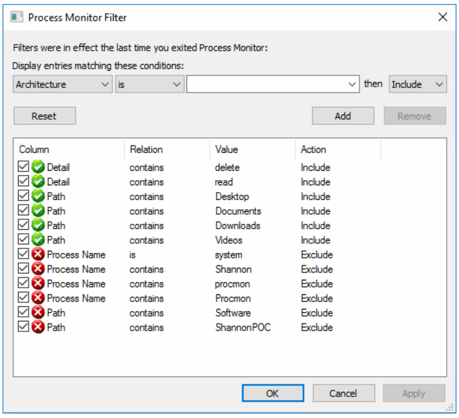

Figure 5.1: Filters enabled while performing test Shannon15

Procmon has been configured to write all the filtered events to a .PML file, which is its own filetype, this can later be converted into a CSV file. Procmon has a command-line-interface (CLI), which was used to control Procmon through C# using the command prompt. It is not a very efficient or elegant method, but it was sufficient for the proof-of-concept implementation. When started, Procmon is constantly logging the wanted file activity, for the honeypot detection method, it is monitoring the honeypots. When the detection method finds a change in the honeypot and deems it necessary to shut down a process it calls Procmon through the command prompt. First, Procmon needs to be shut down in order for it to finish writing the log, this log cannot be accessed before Procmon is properly shut down. Next, Procmon is restarted and begins writing a new log. Through the command prompt, Procmon then parses the PML file into a CSV file, and that file is then parsed into something readable for the shut down program.

Normally no process touch the honeypot files, but once the ransomware has changed the file, several other windows processes might interact with the file as well. These are processes such as Windows search indexer, Windows explorer, system and more. All of these processes will be in the list received from Procmon, these could either be whitelisted or accepted as collateral damage. It was believed that there was no reason for whitelisting since ransomwares could just imitate those process names on the whitelist and avoid the mitigation. Instead, the collateral damage was deemed acceptable. We believe that if more development time was added, the program could be optimized such that the collateral damage could be avoided.

The problem with this method is primarily that it is a third-party implementation, with no easy way of communicating with the program. This results in a very long time in order to find the responsible process and terminating it. But when using C# then the only way to find the process that has changed the file is by using third party methods. Other third party programs have also been considered in this project.

5.2 SSDT

Since almost every system call in the system can be monitored using SSDT, it can also be logged. By having a log of everything that has happened, one can create a pattern and precisely know what files have been hit. Furthermore, once the process responsible for encrypting the files have been found the SSDT can search its log to find what process started the encryption. By doing so the log can show every parent process, every single one of their actions and what files they have created and where. This means that every malicious file stored can be deleted and every malicious process can be killed, including every process started by these processes. Doing all of this would result in a total cleanup of the entire system, covering malicious processes, files and registry changes.

By having control of the SSDT calls, at the same time one can block calls to **vssadmin.exe** in order to prevent the local snapshot from being deleted. By doing so one can create a tiered solution combining SSDT calls and monitoring of **vssadmin.exe** thus stopping the ransomware from encrypting files, killing every responsible process, removing every file and in the end restoring the encrypted files back onto the system.

Chapter 6

Tests

This chapter contains an in-depth explanation of how the testing environment has been developed, including the decisions and challenges leading to the final testing suite. Furthermore, the chapter also describes the test cases designed to test the effectiveness and possible problems with false-positives in relation to the implemented detection and mitigation methods.

6.1 Test environment

A test environment able to test proposed detection and mitigation methods needed to be set up in order to collect the data from the tools created. The primary requirement for the test environment was that it should be able to run the ransomware detection and mitigation tool from inside the environment and collect data from it. Furthermore the system needed to be able to provide the test setup with a new ransomware for each cycle, all of it completely automated.

After looking through several different sandboxing options such as cuckoo [Cuc], it was deemed that none of them fit the specific requirements, due to this, a testing environment was created from scratch.

For the test environment it was decided to use virtual computers through virtualbox. Using vitalization software and taking snapshots allowed the system to quickly revert back to a previous state. Reverting to previous states would be needed after each test of ransomware, to reset the system to before the infection. Virtualbox had the added advantage of being free, opensource and has a well documented command-line-interface.

In total 6 virtual machines was set up:

Quicktester was used to check if a ransomware was active and would work in the test environment.

Baselinetester was used to see how the ransomware behaved on the system, which could later be used to evaluate our tools efficiency.

Testers were made from the last four virtual computers in order to perform parallel testing.

All of the virtual computers were distributed equally among three physical computers. Lastly, the data collection server was a physical computer responsible for storing data sent from the tests and for storing the ransomware such that the test computers had a central base to acquire these from.

The final setup is a series of physical computers running virtual computers used for testing the ransomware. These computers were connected through a network switch which at the same time acted as an access point to the internet. Through the switch they were connected to the data collection server. Giving the test environment its own network setup, ensured a fully segregated network between the development network and test network. It was important that the test environment was able to access the internet to ensure the active ransomwares were able to contact their servers and ensure they performed as they normally would.

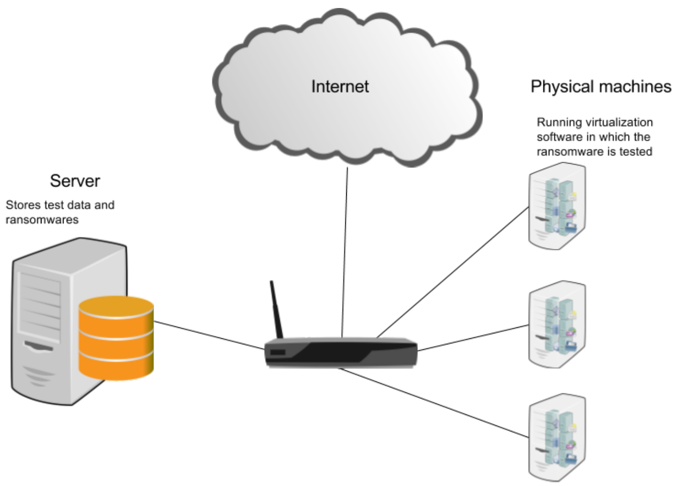

Figure 6.1: Topology of the test environment

To test the effectiveness of the ransomware detection tool, it was decided to test it on actual ransomwares. There are a lot of malware and ransomware repositories online, where researchers can acquire them. A collection of 38,152 crypto-ransomware from 2013 to July 2016 was downloaded from VirusShare. Later we found out that a lot these were no longer active or not binary executables which was needed for testing. Another 69 were manually acquired primarily from rəverse.it which were recent ransomwares such as WannaCry as executable binaries, theZoo on Github and was also used. This made it possible to have a wide range of ransomwares, from the beginning till may 2017, see section 7.1 for a deeper analysis of the tested ransomwares and distribution of the families.

In order to avoid wasting resources on testing inactive ransomwares, and ransomwares that would not work in the test environment due to either not being able to be executed or due to anti-analysis techniques employed by them as described in section 2. A preliminary analysis was performed on the ransomwares before the actual test against the proof-of-concept detection methods. The preliminary analysis consisted of two tests on each ransomware, a coarse grained test by our Quicktester, and a fine grained tests to further remove non-working ransomwares. The preliminary analysis managed to test 6.393, and after it, 65 ransomwares remained that could be considered active in the test environment.

The advantage of the designed test environment was that it was rather easy to ensure the ransomwares did not spread uncontrolled through the network. Furthermore since, the data collection server was running Linux and all ransomwares had their extensions removed, accidental execution of the ransomwares was not possible. Another advantage is that sending the stored information over the network allowed us to collect it centrally right away, without the possible implementation problems of having to directly extract the information through the virtual computer. A typical flow is:

1. Host controller starts the virtual computer

2. A specially designed program then contacts the Datacollection server to request what ransomware it should work on. Which is then, downloaded from the server over FTP and executed.

3. While the ransomware is running, data is collected and stored locally, such as files affected, resource usage and more. It is also during this step the ransomware is supposed to be detected and mitigated.

4. 25 minutes after the ransomware was started, the specially designed program, takes a status of the system, identifies all the changes the ransomware made and posts it all to the server, through an API written in PHP. The information is stored in a MySQL database on the server.

5. The host controller registers that the data has been posted and reverts the virtual computer back to before the ransomware, and the cycle start over. If any issues arise on the virtual computer such as a bug in the software, crash or the ransomware in some way prohibits it from sending the required data, then the host controller has an upper time limit, and once reached will restart the cycle automatically.

When it started to work, it was very efficient since everything was completely automated, the only thing that needed replacement from time to time was the detection and mitigation software. However, this type of environment had a lot of problems due to segmentation between test and development environment. Debugging program errors was very tedious, as everything had to be run from virtual computers, it was not possible to properly test programming changes before deployment. After any change, committing and synchronizing the changes was necessary. Once the files were ready for deployment they had to be added and several new snapshots of the virtual machines had to be taken to ensure revertability. This resulted in, any coding change took at least 20 minutes to implement. In some cases, it was necessary to use a different version of the testing machine such that debugging the applications through Visual Studio live, while the ransomware would attack the system, was possible. This resulted in a growing amount of snapshots, resulting in problems with storage capacity which would sometimes lock down the whole testing environment.

6.2 Data collection server

The data collection server was the primary data storage server. It was responsible for storing all data from the tests in a database, and providing the tests with the relevant ransomwares.

The server was an Ubuntu Desktop 16.04 LTS, running FTP, Apache, slim framework, PHP and MySQL. The hardware of the computer can be found in appendix B. Apache, slim and PHP was used to allow the tests to communicate with MySQL. An API was implemented on it, so the tests could contact the server to acquire the ransomware and to post data. A more detailed explanation of the API can be found in section 6.2.1. All the test data was stored in the same database, but separated into different tables for each test case. Even though a lot of precautions were taken to avoid accidental infection of ransomware, the database was backed up every night and stored in Dropbox. Even if the Dropbox folder would be hit, then Dropbox has revision control allowing us to restore any encrypted files. A more detailed explanation of the database along with the defined tables and their rows can be found in section 6.2.2. The FTP server was used as a simple way for the test machines to download the ransomware. All the ransomwares were located in the same folder which was shared through FTP. Once the tests had acquired the name of the ransomware to perform tests on, it could be downloaded from the server.

6.2.1 API

The programs developed for testing the ransomwares, including both the software running on the virtual computers and the physical computers, communicated with the data collection server using standard CRUD operations (Create, Read, Update, Delete) implemented in PHP, although delete was not possible in the designed setup.

It was important to use an API to ensure that there was no direct link between the infected machines and the database storing all the data in the case some of the ransomwares would target our database, as has been seen before[Mag] [Tec]. Originally, for simplification all requests to the server was shaped as GET request, even when posting data to the server (even though this is not best-practice). When requesting the name of the ransomware to work on the request would look like this:

```
http://192.168.8.102/v1/index.php/getbaseransomware
```

When informing the server that the ransomware had been downloaded a POST masqueraded as a GET request was sent, in the following form:

```
http://192.168.8.102/v1/index.php/
                        postbasefetched?RansomwareName=CryptoWall
```

The original idea was, that this type of implementation would be faster since it avoided having to define headers and request bodies, and it would still be sufficient for the needs.

However during testing a problem with posting was noticed with all of the information collected through the browser. The problem resided in generating an URL that was too long. Some of the data we collected was fullpath of all files changed, which could be more than 30.000 file observations. Each of which would usually be more than 30 characters, resulting in at least 900.000 characters, and this was just for one of the parameters collected. According to research performed by Boutell[Bou] most browsers does not support anywhere near such long URLs, and best-practice also dictates to avoid URLs longer than 2.000 characters. This lead to reprogram parts of the API, such that in cases where it was needed to post a lot of data, the actual POST operation with correct headers and data stored in the body was used instead.

There were 7 API calls used by our testing environment.

getbaseransomware: This one was used by the primary logging software to identify the ransomware needed, and thereafter download it from the server.

postbasefetched: As soon as the ransomware had been downloaded to the system, this API call was performed, and a timestamp was inserted in the database. This made it possible to track when ransomwares were downloaded.

postbasetaken: This one was used to ensure knowledge of what ransomware had been taken, and is used by the getbaseransomware to identify and return the correct ransomware.

postbasestarted: Once the ransomware is downloaded, the next step is to execute it. When it has been executed, this method is called, and another timestamp added, such that it can track when the ransomware started which is used for data analysis.

getbasehost: This method returns the ransomware currently missing a posted timestamp, meaning the test has not yet finished. This allows the host controller running on the physical computer to continuously ping the server, to check whether the test has completed. Once the information has been posted to the server the host controller can restart the virtual computer and the test cycle starts over on a new ransomware.

postbaseposted: This method is a POST request which is different from all other API calls that are GET requests. This posts all of the information gathered by the program running on the virtual machine and is usually several megabytes in size.

postbasetested: Once everything has been successfully posted, this method is called and sets a flag to true in a column on the database. This was primarily used for debugging.

In appendix D, parts of the source code for the PHP code can be found.

6.2.2 MySQL database

Just like the API have dedicated API calls for each tests, so does the database which contains a table for each test case. Most of these tables were identical, but in total there were 3 different kinds. One type for the Quicktester, one type for the Baselinetester and one type for all other tests.

The Quicktester table had 6 columns. The first column, which also counted as the primary key, was the RansomwareName. The Quicktester table started out with being populated with all of the ransomware names, consisting of 38.220 rows. It also had 2 timestamp columns, one for when the ransomware was downloaded, and one for when data was posted. Unlike, the others, this table did not contain a timestamp of when the ransomware was started. This information was not relevant in the Quicktester, as it only needed to verify that the ransomware was active and would work in our test environment. Furthermore a column containing a boolean value called 'active' was also present. This was used to mark ransomwares as either active (1) or inactive (0). Lastly, the columns "TakenBy-Baseline" and "TestedByBaseline" were used by the Baselinetester to identify what ransomwares were currently being tested, and which ones had completed testing.

Similarly to the Quicktester, the Baselinetester also contained the columns, "RansomwareName", "Fetched" and "Posted", however, besides these the Baselinetester had an additional 16 columns for storing data about how the ransomware affected the system. The data gathered and stored was information such as amount of new files created, files deleted, percent of the hardware resources used such as RAM, CPU and the disk. Furthermore the complete path to all of the changed, deleted and new files were gathered and stored. These could be several megabytes in size for each category, so the columns were designed to be of type *longtext*, resulting in them being able to store 4 gigabytes of data. This is much more than needed, however the other option would be a *mediumtext* which is limited to 16 megabytes, which was believed to be too little, in case some of the tests contained significantly more data.

Finally, an additional 2 columns for each ransomware test were stored in this table, *TakenByX* and *TestedByX*. Just like the similar columns from the Quicktester, these were used by the different tests to identify how far in the testing process all the ransomwares were and helped to keep track of this.

The final count of columns in the baseline table was $19 + (n * 2)$ where n is the amount of tests performed, rendering a total of 35 columns.

The table for all other ransomware tests were very similar to that of the Baselinetester except, instead of having the control columns *TakenByX* and *TestedByX* they had information directly related to the ransomware. Firstly, column *NameOfShutdownRansomware*, contained a list of all the processes that were identified as being malicious and shutdown by the mitigation solution. This would help identifying possible false-positives, such as incorrectly shutting down e.g. explorer.exe. Furthermore, since there is a substantial delay between detection and mitigation due to the way the process performing a specific action is identified, two additional columns containing timestamps has been added, one for when the malicious activity has been detected, and one for when processes has been stopped and killed.

In Appendix C, the database structure for all 3, including all of the column types can be seen.

6.3 Test computers

The part of the test environment that was responsible for testing the ransomware consisted of three physical computers and six virtual computers. These were configured such that two of the physical computers were identical, however because of limited resources one of the physical machines was different that the others. The virtual computers were identical. The hardware specifications can be seen in appendix B.

The physical computers was a standard Windows 10 Enterprise install, with updates and sleep function disabled, to ensure the test environment did not shut down during testing. The software specifications can be seen in appendix B. The physical computers also known as host computers, had two functions. The first being running the virtual computers where the actual testing was performed, and secondly to restart the test cycle when the ransomware had been tested or when a certain timer had passed, marking the test as failed.

The virtual computers were divided into three different types, Quicktester, Baselinetester and Tester. They all shared the same basic setup in relation to hardware and software.

- 2048 MB RAM
- 1 virtual CPU
- 50 GB Harddisk

- 1 shared folder between virtual computer and physical computer, acting as a read only network drive.

The virtual computers also had Windows 10 Enterprise installed, however setup deviated slightly from a fresh install and from best practice. This was to ensure as few parameters as possible affecting the results. The computer was thus configured such that:

- Windows update was set to notify, but not download nor install. This was done through local group policies, using gpedit.msc.
- Automatic login was setup using the netplwiz service, although later the usage was not important, since the testing cycles were restarted from an already booted machine
- Windows defender was disabled, to ensure old known ransomwares would not be caught by it. This was done trough local group policies using gpedit.msc.
- User-Account-Control was *"disabled"* by setting it to the lowest security, i.e. never notifying.

To make the test system seem like a real system, a set of common programs [Cle] were installed along with a set of dummy files to populate the system. The dummy files will be explained in detail in section 6.5. Examples of the installed software was, Google Chrome, Java Runtime Environment, .Net Framework, FoxitReader and more, a complete list, of programs, their versions and Windows updates can be found in appendix B

Once all the above steps was done, the virtual computers were ready for testing. Originally the relevant software was added to the startup folder so it would automatically start when booted. However, due to problems with the program not properly starting with administrative rights, because of Windows constraints, it was revised and changed to being started from Windows Task Scheduler, and scheduled to start on login. Which was again later revised, due to ransomwares encrypting parts of the program's needed dll files and problems with Windows Search Indexer slowing the system down by a factor 10 after boot. The final configuration was storing the ransomware detection and mitigation software in the *sysWOW64* folder within the Windows system folder. The software was executed, and within it a sleep timer of 30 seconds would make the program execution wait. During this wait, a snapshot of the system was taken, then the host controller would for each test cycle, revert back to this stage. This way it was not needed to make hacks to ensure the program started with administrative rights at boot, since it could just be run with admin privileges.

6.4 Liveness tests and data collection

As mentioned earlier, we had 3 different types of virtual computers, collecting slightly different information.

The Quicktester was made in order to quickly determine if a downloaded malware was an active ransomware or not. This was needed due to having a data set of more than 38.220 different malware. The Quicktester ran a malware, monitored the system, and if more than five files changed during five minutes, then the malware would be considered an active ransomware. This means that there is a high chance of false positives and false negatives. If a real ransomware uses more than five minutes before starting then it would not be considered a ransomware by the Quicktester. Likewise if a malware, that is not a ransomware, tampers with more than five files then that would be considered ransomware. The purpose of doing this preliminary sorting is that a quick test takes a maximum of five minutes, where the tests of the Baselinetester and different detection methods can take up to 80 minutes.

Once, a change had been detected and deemed to be due to ransomware, the ransomware was considered active and the information was sent to the Datacollection server. The host controller for the Quicktester, located upon the physical computer, would notice that information had been sent, and restart the test cycle. This cycle had an upper limit of 5 minutes. If nothing happened within that time, either the ransomware did not execute or the ransomware had some sort of dormant period. In either case, it would not be fitting for further tests and therefore marked as not active. Long dormant periods can not be used as it would make the later tests run for much longer, which would not be feasible.

The Baseline tester worked in part to further detect if a ransomware was active, but also to collect information about each ransomware to get a perspective for the tests using the detection and mitigation tool. Similarly to the Quicktester, file system changes were identified using the file monitor, however this time they were also stored along with a timestamp of when the changes occurred. This was used as tools to later analyze in what order the ransomwares encrypt files or other possible patterns. Furthermore, after 25 minutes, the Baselinetester took a hash of all files included in the test suite, this was compared to hash taken before the execution of the ransomware. This way, we could easily see the total amount of test suite files affected by the ransomware, which allowed for an easy comparison between detection methods and their effectiveness.

Lastly, the ransomware tests were where the effectivity of detection and mitigation was measured. They collected almost the same information as the Baselinetester with the addition of a timestamp for when the ransowmare was detected, a timestamp for when the first process identified as ransowmare was shutdown, and a list of all processes shutdown.

6.5 Test cases

For the tests, a set of test files called the *"test suite"* was created, it consisted of auto-generated .doc documents based on Harry Potter and the Philosopher's Stone. Along with that, documents from The Technical University of Denmark's intranet *CampusNet* were downloaded and included. This gave a diverse file portfolio consisting of a lot of common files types, such as word documents, pdf and pictures, ranging in size from a few kilobytes to hundreds of megabytes.

Originally, the idea was to test the whole file system, however due to difficulties with properly implementing such a system, it was decided to look at 4 selected folders instead. The selection of the 4 folders should not have an influence on the actual test data since when comparing the test results they are compared based on the amount of files the ransomware manages to encrypt before being detected and the system reacts. The 4 folders were *"Desktop"*, *"Documents"*, *"Downloads"* and *"Videos"*, all within the users directory.

A similar structure of the directories was desired, but without having to duplicate files across the folders. The similarity was needed in order to compare the directories when determining ransomware encryption patterns. To do this, the program FolderSize [Siz] was used, because it made it possible to analyze the files in the test suite, which enables us to see the size and amount of files. Furthermore, a tool was developed to help analyze the placement in alphabetic order and meta data of the files, such as creation timestamp and last modified timestamp. The tool iterated though every file in the four folders and found relevant properties of these files, this was done such that when comparing this data with the test results it was possible to find patterns between encryption order and file properties.

Each of the detection methods below, have four different test cases, where one parameter has been tweaked between each test. Each cycle of testing of ransomwares took between 30-80 minutes. A normal run would be starting up, running some preliminary ransomware executions checks, then executing the ransomware. While running, data is collected and an attempt is made to detect and shutdown the ransomware. After 25 minutes, the program enters the ending phase, which is where it collects information about what files have been altered, such as changed or deleted. It also sorts the information and prepares to send the data to the server. This last part can take anywhere from a few seconds to several minutes depending on the information gathered. Usually this type of cycle takes roughly 30 minutes to complete, however, in some cases the preparing of the information took longer, thus a higher upper limit was needed. The upper limit of 80 minutes is enforced by the host controller which shuts down the test and starts a new test cycle. Usually the upper limit of 80 minutes would be reached due to the testing software crashing or in other ways having issues resulting in a missing post to the database. In most of these cases, it would be sufficient to restart the test of the specific ransomware.

6.5.1 Honeypots

As explained in section 4.1, the idea is to let specific files be monitored by the detection software for changes. In the case of changes, the responsible process is flagged as ransomware. The process name is acquired using the third-party software Procmon, and then shutdown.

In theory, one could have 99% of all files on a system to be honeypot files, this would ensure en extremely high success rate, however, this would also take up a lot of hard disk space, which makes the concept unfeasible. Furthermore this would never be the case for any normal system. Instead, one should base the effectiveness of a few honeypot files, which is the case in this project.

For the honeypots, there are four different setups, each one with a different honeypot to files ratio.

- Test 1: This test had 29 honeypot files out of 2.887, or 1,001% of the files. These files covered the file types; .docx, .jpg, .xlsx, .pdf, .pptx filetypes. In the size range 10,5KB - 15,4MB.
- Test 2: This test had 63 honeypot files out of 2.921, or 2,157% of the files. These files covered the file types; docx, .jpg, .xlsx, .pdf, .pptx, .php, .mp4, .xls filetypes. In the size range 0,5KB - 106MB.
- Test 3: This test had 161 honeypot files out of 3.019, or 5,333% of the files. These files covered the file types; docx, .jpg, .xlsx, .pdf, .pptx, .php, .mp4, .xls, .zip, .txt, .java, .cpp filetypes. In the size range 0,25KB - 154,9MB.
- Test 4: This test had 305 honeypot files out of 3.163, or 9,643% of the files. These files covered the files types; docx, .jpg, .xlsx, .pdf, .pptx, .php, .mp4, .xls, .zip, .txt, .java, .cpp, .doc, . filetypes. In the size range 0,25KB - 154,9MB.

The honeypots in Test 1 were also present in Test 2 with the addition of 63 new files, just like the honeypots present in Test 2 was present in Test 3 and so on. It is important to note, that except for the file type, the files were selected at random and added at random to the 4 folders. When the honeypots were created there was no knowledge of which order the ransomwares encrypted files in, and the theory was that it would be different from ransomware to ransomware and therefore as long as the tests had the same files, they would be comparable.

6.5.1.1 False-positives

In order for the detection method to be viable for actual use, it should have as few false-positive reactions as possible.

Since this detection method uses honeypots, which no program or user will normally interact with, it is fairly easy ensure few or no false-positives. Any process tampering with the honeypots are considered malicious, but for the sake of the case where the user might modify the file by accident, a threshold of 2 was set. Such that if 2 or more of the honeypot files were modified and changed within 1 minute, then it is unlikely that it was the user, and thus the process was shutdown as it is classified as malicious.

Due to this threshold, no actual false-positive test was performed, since the test would have to be defined as to how often the user would interact with it. However, testing for the best way to avoid users or programs accidentally interacting with the files could be defined. This was deemed out of scope for the project.

6.5.2 Shannon entropy

The shannon entropy detection method has been implemented as described in section 4.6.2. To decide whether the new entropy of a file that has changed is suspicious, a test is made for analysis. First, the entropy of all files in every directory of a non-encrypted system was made. Next the same system was encrypted by a ransomware, and the file entropy was then recalculated for every file. The ransomware for this test encrypted the files and added a *.fun* extension upon the file, which made it easy to know what file was changed.

Thereafter the entropy of the files pre and post ransomware encryption were compared in order to know how much the entropy changes when a file is encrypted. Since the entropy varies between 0 and 1, it is hard for a file with entropy **0.99** to have a high rise in entropy whereas for a file with entropy **0.2** it can have a much higher increase in entropy. Therefore the different files were divided into several different categories based on the files entropy before encryption. The first nine categories are with **0.1** interval in original entropy, such that the first category is from **0.0** to **0.1** the next from **0.1** to **0.2** and so on. After **0.9** it changes such that the interval is **0.01** and after **0.99** the next interval was to **0.999**, **0.9999** and last to **1**. A full list of the interval categories can be found in appendix E.4.3.

By doing so, different changes in entropy would be deemed suspicious for different files. If the same change rate were to be suspicious for every file, the low entropy files would have very low tolerance for changes whereas high entropy files would have a very high tolerance. For example, files with original entropy between **0.5** and **0.6** has an average increase in entropy by **0.29** when encrypted, where the files with entropy between **0.95** and **0.96** has an average of **0.04** higher when encrypted.

Now the increase in entropy deciding whether the change is deemed suspicious and what needs to be added to the threshold is known.

The four different versions of the shannon entropy detection system that has been made is based upon the value of this threshold. Once the threshold is reached the system reacts and shuts down the process. To trigger this reaction the threshold must be reached within a minute, otherwise the trigger does not count toward the threshold. This variable could be altered depending on detection method, but the results will be clear with the change of the threshold only. The different amount of suspicious actions in order to reach the different thresholds are 3, 5, 10 and 15.

Naturally a version with a lower threshold will detect a given ransomware quicker. The tests are not made to see which one is best, rather to see how big the change is between the different thresholds. A lower threshold means a higher probability of having a false positive, therefore it is desired to know how damaging a high threshold is to detection and mitigation performance.

As mentioned in section 4.6.2 the shannon detection method needs to read every byte in a file once there have been a change to that file. If that file is locked by a ransomware or some other program then it is not possible to get the bytes of the file. This makes a ransomware that locks the files after encryption able to avoid detection from this method.

6.5.2.1 False-positives

The possibility for this detection method to wrongly assume that a ransomware encryption is in progress is unfortunately quite high. Since PDF files have a natural high entropy, the detection method would react if a large number of PDF files suddenly were to be copied into the system. Not only PDF files, but if a large number of high entropy files were to be copied into the machine from an external drive or similar, the detection method would also react. The threshold set in the detection methods cannot, unless dedicated work is made, be reached naturally without adding files from outside the machine. No user would make large enough changes to change the entropy such that it causes suspicion, in 5 files within a minute. Since this is highly unlikely, this detection method is still reliable enough when it comes to false positives.

A false positive test was made in order to check how the detection method reacted to normal use of a system. First a game called Hearthstone was installed upon the system, this triggered several reactions from the detection method and also caused it to crash. The reaction happened due to the game installation created several temporary files that often changed, these files also had high entropy. The crash of the process running the detection method were due to unforeseen errors in the code only triggered when editing a file several times within a second. This crash might indicate why many of the shannon entropy tests came back without any results. An installation of Open Office was made as well, this did not cause a crash of the tested process but still triggered several reactions from the detection method.

Simple actions upon the system was tested after the installation tests. Several files and folders were deleted in order to test if that would trigger reactions, which it did not.

Copying files from an external directory into the system did, as expected, cause reactions from the system, also copying from one folder to another, both in the system. Compressing a folder with zip also triggered a reaction, but only a single reaction, meaning that if the user does not create more than 1 zip file within a minute, than it is below the threshold and then it will not react.

Chapter 7

Analysis and Evaluation

This chapter first presents the test results obtained by the different detection methods, these test results are then discussed and analyzed. Following this are the different ransomwares analyzed, this includes their encryption pattern and other distinctive features. Lastly is an analysis of ransomware using game theory.

7.1 Data analysis

The data from the many tests made has been gathered into readable and understandable plots in order to show the performance of the different methods detected a ransomware. The most important aspects for the methods are speed and efficiency, meaning how many of the ransomwares are successfully detected.

The performance of the different detection methods is shown in figure 7.1. They have been tested on 65 different ransomwares, but some of the tests did not provide any data, as shown in appendix A.1.4. The test that did not provide any data is due to various reasons, sometimes it is that the ransomware terminates the detection method, thus the program logging the activities made by the ransomware, other times it is due to an unforeseen error in the detection method. Appendix A.1.4 also shows the performance of the different test methods as pictured in figure 7.1.

Figure 7.1 sharply shows the success rate of the different detection methods, clearly the honeypot detection methods have a much better detection rate than those using shannon entropy. This figure does not disclose information about whether the mitigation of the ransomware has been successful, only that the presence of ransomware was detected.

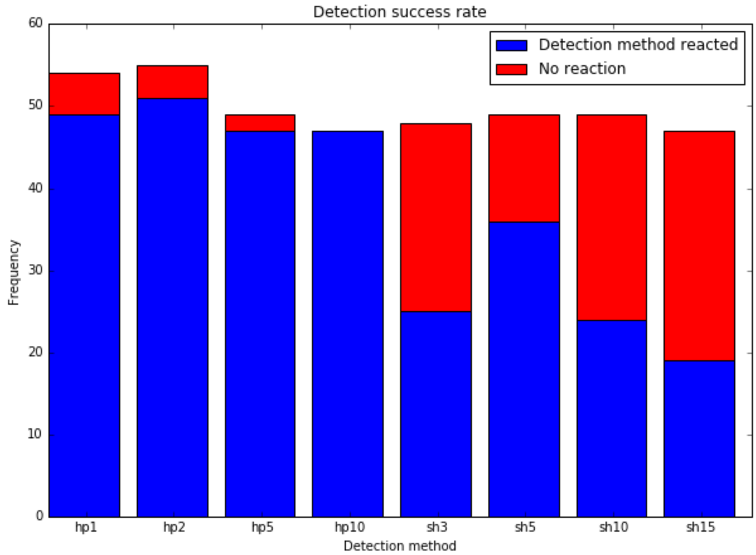

Figure 7.1: Detection success rate

One of the reasons for the low detection rate in shannon is due to the hardcoded variables that determine how much a file needs to change in order to be suspicious as found in appendix E.4.3. The few ransomwares that avoid detection from the honeypots might be due to a lack of honeypots encrypted within the specific timeframe. As seen in appendix A.1.4 many of the ransomwares that hp1 or hp2 does not detect has not gained any results from the remaining test methods, indicating that it could be a ransomware that has methods to counter detection tools.

In figure 7.1 we have shown whether the detection methods are able to detect ransomwares, however, the speed at which ransomwares are detected, is also important. This can be represented in several different ways, the first option chosen as a representation is the total number of files the ransomware has encrypted. We assume that the encryption method and speed of encryption is nearly the same through every test method.
As seen in figure 7.2 the files encrypted by the ransomware is represented in boxplots. This clearly shows that hp5 and hp10 is more effective than hp1 and hp2 as it was intended. The fact that hp10 is looking a bit slower than hp5 will be discussed later. Shannon entropy as shown, is much less effective than the honeypots, this is partially due to the efficiency of the ransomware as shown

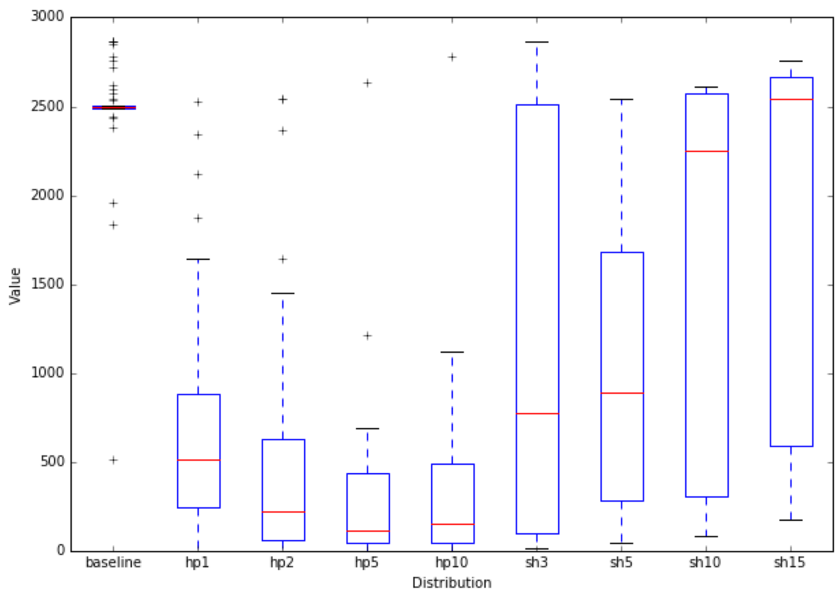

Figure 7.2: Files encrypted by ransomwares

in figure 7.1 and might be a result of unfortunate shutdowns of the detection method or a flawed method of mitigation.

The outliers for baseline in figure 7.2 is because they have targeted less file types and possibly because the ransomware has a slow encryption. The types of files that the ransomwares encrypts is shown in appendix A.1.3.

The other way of measuring the detection speed is much more direct, instead of looking at how many files that has been encrypted by the ransomware in the test, we look at the time from the ransomware is executed until the detection method first detects a suspicious process.

Figure 7.3 shows the time it takes to detect the ransomware from its execution. Some ransomwares have built-in delays before encrypting files, others start right away, this varies. In this boxplot all of the honeypot detection methods roughly have the same detection time, hp2 being the absolute fastest with a median detection time of 47 seconds. Comparatively the shannon entropy detections have very different time spans from start to detection. Theoretically speaking

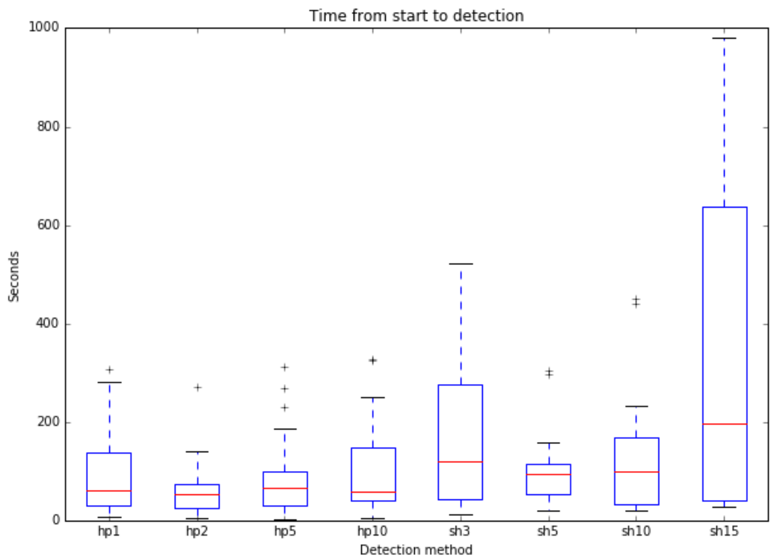

Figure 7.3: Time from ransomware start to first detected

the shannon with the lowest threshold, sh3, should be the fastest followed by sh5 and so on, which is explained later.

After a thorough analysis of the data, it has been determined that even though the virtual machines that ran the test had the same setup, the physical machine seems to have affected the test methods. This is shown in figure 7.4. The boxplot shows the time it takes for the program from detection to shutdown of the ransomware. As written in section 5.1, the shutdown of a ransomware is slow because of using third party programs. The shutdown time should however, have been almost the same for each detection method.

The boxplot shown in figure 7.4 shows that there is a big difference in the time it takes to shut down a ransomware. The detecion methods were distributed across the different physical machines as following:

Computer 1: baseline, sh3

Computer 2: hp1, hp2, sh5, sh10

Computer 3: hp5, hp10, sh15

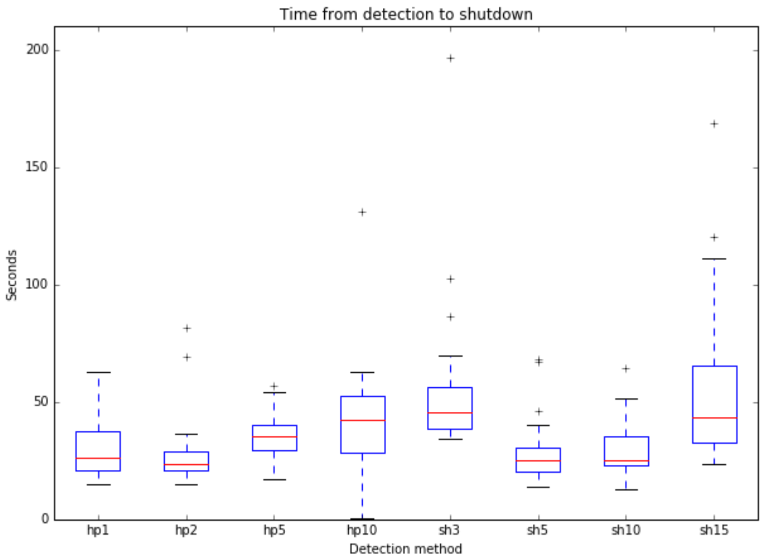

Figure 7.4: Time from detection of ransomware to assumed shutdown

Computer 1 and computer 2 are identical whereas computer 3 has a SSD hard disc and a less powerful CPU. Furthermore, it also has more RAM, the full hardware list can be found in appendix B. The difference in computers is clearly shown in the data obtained in figure 7.4. We believe that the physical hardware difference have had a significant impact on our test results. Since computer 1 and computer 2 are identical in hardware yet still have a clear difference in their test, we are inclined to think that there must be some unknown variable affecting our results, particularly sh3.

Computer 2 has a CPU that is 68% faster than the CPU in computer 3 [Int]. Therefore, by adjusting the time from detection to shutdown for hp5, hp10 and sh15 it should show whether the hypothesis is correct. The adjusted result can be seen in figure 7.5. This is still not exactly the same, but it is closer to the theoretical result. Why the results from computer 1 and computer 2 are as different is, as previously stated, unknown.

In figure 7.2 it is shown that hp10 lets the ransomware encrypt a few more files than hp5, the reason for this could be that hp10 has a slower shutdown than hp5, and why that could be is unknown. We assume that the optimal amount of honeypots is between 5% and 10% of the total files upon the computer based upon the results from hp5 and hp10. An analysis of the optimal placement of honeypot files is given in section 7.3. The optimal solution for the shannon entropy is hard to determine from these test results and requires further testing in order to give a definitive answer.

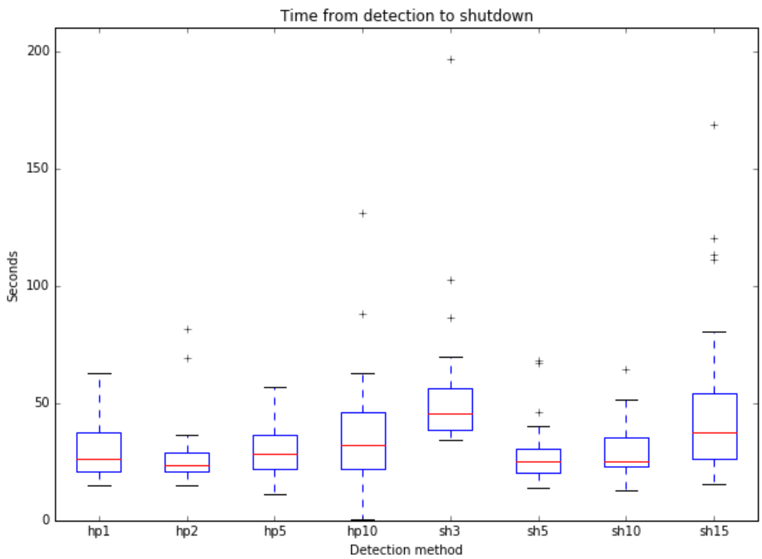

Figure 7.5: Normalized figure 7.4 with the CPU specifications of computer 3

7.2 Ransomware analysis

The actions of the 65 different ransomwares has been monitored in an environment where no process attempted to stop them. This was done in order to determine what pattern the ransomware used to encrypt the files, what file types were targeted by the ransomware and monitoring of the hardware usage.

The sha1 value of the different ransomwares are represented in appendix A.1.1 along with their an alias.

In order to find the pattern of encryption a file monitor was monitoring the entire user directory and logging every change to files in this directory, this data was then parsed to something easily readable and analyzed to determine the pattern which the ransomware encrypted the files in. The data can be found here and the results can be seen in appendix A.1.2.

The different encryption patterns found are:

1. Alphabetically, all files in the current directory are targeted first, then following the same procedure in sub-directories.
2. Alphabetically, everything is taken alphabetically, including sub-directories, meaning the ransomware starts in Desktop, encrypts the files from a to e, then target a sub-directory that starts with f, and after that directory is done, continue with files g-z in the desktop directory.
3. Alphabetically, with the directories in reverse, meaning it starts in the videos directory and targets a sub-directory there that starts with v, but then encrypts all files in that folder alphabetically.
4. Like the second encryption pattern, only the ransomware creates a long path of directories in the current directory that in the end stores a .txt file. The filepath is for example

   ```
   C:/Users/Baseline/Desktop/u00ca/u00c0/
   u00ca/u00d0/u00c0/u00d1/u00d8/u00c8/u00d4/u00d0/u00ce/u00c2/
   u00c0/u00d2/u00dc/u00d4/u00c0/u00c9/u00cb/u00db.txt
   ```

5. This encryption pattern seems random. Only occurs in R56.

Almost every single encryption pattern is alphabetical in some way. This means that for encryption pattern 1, 2 and 4, a honeypot placement method, where the honeypots are named such that they are the first files alphabetically, would be the optimal honeypot placement for these patterns. Since only three of the tested ransomwares have a non-alphabetical pattern, this would mean that the majority of the ransomwares would be detected much faster. However we believe that future ransomware will have a random encryption pattern, as concluded in our analysis in section 7.3.

The ransomwares has been analyzed by rəverse.it in order to gain information about ransomware type and whether or not the ransomware deletes the VSS, mentioned in section 3.4. The date provided in appendix A.1.2 is the date that, that particular file has been submitted to either VirusTotal or Metadefender for analysis. By looking at appendix A.1.2 it can be seen that no ransomware up until February 2015 actually deletes VSS. This might be due to the new families of ransomware such as CryptoLocker and TeslaCrypt.

The file types encrypted by the ransomwares have been found by looking at the files saved in the log files of the test. These file types vary between the different ransomwares and can be seen in appendix A.1.3. As the data is obtained from the test data, the file types are limited to the different filetypes that exists in the testing environment.

Appendix A.1.4 shows that all of the detection methods are unable to return any data from some tests. This is either due to a program crash or the ransomware has some countermeasures against being shut down. Ransomware R56 and R62 are great examples of ransomwares that might not be straight forward to mitigate. That being said, the baseline has run tests upon these ransomwares and been able to get a result returned, therefore the problem might accour when the detection method tries to mitigate the ransomware.

Using rəverse.it to analyze the different ransomware files, their type has been estimated by rəverse.it. Appendix A.1.2 shows that there are multiple different ransomwares for the tests, but also some that are repetitive, such as Xorist and TeslaCrypt. If there are two names in the namespace then it is due to the fact that some antiviruses identify the ransomware differently. These ransomwares span over a broad timeline, from 2011 to 2017 whereas the distribution is quite fair.

One of the reasons that the number of Xorist is quite high is that Xorist is not a standard ransomware. This ransomware has a builder for Xorist ransomwares which the creator of the ransomware has sold to people such that they could create their own version of an Xorist ransomware [Cim].

7.3 Game Theory applied on Ransomware

Game Theory in relation to ransomware, can be divided into two cases:

1. Two-player game between the cyber criminal and the victim
2. Two-player game between a ransomware and an anti-ransomware software

Using the theory presented in section 3.5, we can analyze the cases. Case 1 can be considered as a non-cooperate, non-zero-sum, dynamic game with complete and perfect information. In figure 7.6 the extensive normal form game can be seen.

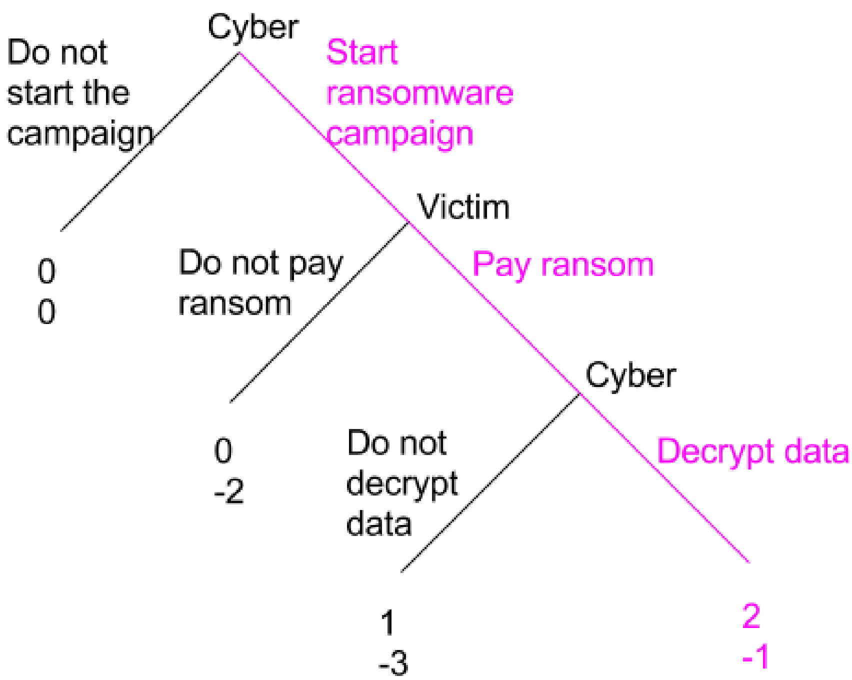

Figure 7.6: Extensive normal form game representation showing optimal solution

This game can be solved using backwards induction. We begin from the cyber criminals second move i.e the third stage. Here he can choose to either decrypt with a payoff of 2 or not to decrypt with a payoff of 1, so to decrypt is the optimal choice. This means that at the second stage the victim anticipates that if the game reaches the third stage, the cyber criminal will choose to decrypt the data, resulting in a payoff of -1. So at the second stage, the victim can either choose to pay the ransom with an expected payoff of -1 or not pay the ransom with an expected payoff of -2, which means the victims best-response is to pay the ransom. This leaves us with analyzing the first stage, where the cyber criminal can anticipate that if second stage is reached then the victim will choose to pay, resulting in a payoff of 2. Thus at the first stage, the cyber criminal can choose between not starting the ransomware campaign with a payoff of 0, or starting it, with a payoff of 2, which means starting the ransomware campaign is the optimal play.

The conclusion is that it always pays off for cyber criminals to start ransomware campaigns, and for the victims to pay, since the criminal will decrypt their files. Which is also indicated by the purple marking.

It could be argued that if the victim pays, then the payoff for the cyber criminal is the same no matter if they decrypt the files or not. However, we would argue that, the payoff for decrypting is higher. This is partly due to the fact that if the cyber criminal decrypts the files, they create an incentive to pay. If victims rarely get their files back, then they would be less likely to pay the ransom, so it is the interest of the cyber criminal to decrypt the files.

Furthermore, the payoff is designated for the victim, under the assumption that the victim does not have proper backup if any at all, and there are no publicly available decryption tools. Of course this is a simplified version of the real world. However, data about this is significantly lacking, and how individuals and companies value their files depend on which files are lost and the ability to recover them. If every organization and individual that could be hit, had a full backup, then the tree would look similar to the one shown in figure 7.7, and when analyzed it becomes clear that at the victims first choice, would be to not pay the ransom since it has the highest payoff. And since the game is of complete and perfect information, the cyber criminal would know this to be the optimal play of the victim, and therefore they would be equally likely to either start the ransomware campaign as they would not, since the payoff in both is 0. It could be argued, that if the cyber criminal has a payoff slightly lower than 0 in starting a ransomware campaign since it does require some resources, which means not to start the ransomware campaign is the optimal choice.

Case 2 was the game between the ransowmare, and the anti-ransomware. This can be considered as a static game with complete information. The analysis in section 7.2 showed that there were three primary methods for encrypting, either in alphabetical, reverse alphabetical, or random order. To find the optimal

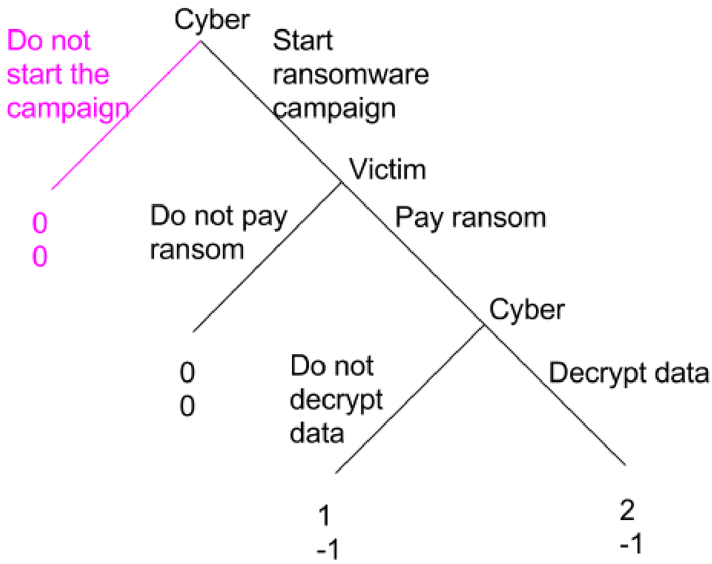

Figure 7.7: Optimal solution if everyone had complete backup

placement of honeypot files according game theory, we have constructed the figure seen in 7.8

For this case, let's say we have 100 files and one of them is a honeypot file. If the anti-ransomware solution places 1 honeypot file as the alphabetically first file, and the ransomware also is alphabetical, then the payoff for both is 0. If however, the ransomware works in reverse alphabetical order, it would encrypt all 100 files before being noticed. If either works in random order then the value has been set to 50 which is the number of files that can be encrypted before there is more than 50% of hitting a honeypot. The best-response for the ransomware would therefore be to work in opposite order of the anti-ransomware. However, since the ransomware, does not know in which which way the honeypot files are distributed on the system, it will have to assume that the anti-ransomware places them based on its best-response. The anti-ransomwares best-response is to work in the same order as the ransomware, but again, it does not know in which order the ransomware encrypts the files. As seen in figure 7.8, both the ransomware, and the anti-ransomware has a best-response when playing (Random, Random) therefore there is a Nash Equilibrium with the resulting payoffs (50,-50).

Ransomware/ Anti-ransomware	Alphabetical	Reverse alphabetical	Random
Alphabetical	0/0	100/-100	50/-50
Reverse alphabetical	100/-100	0/0	50/-50
Random	50/-50	50/-50	50/-50

Figure 7.8: Normal form representation with best-response and Nash Equilibrium indicated.

Thus, we can see that it is highly likely that as anti-ransomware solutions become more common, that ransomwares will start to encrypt files at random, and anti-ransomware solutions using honeypots, would place these files at random in each directory.

Chapter 8

Conclusion

The purpose of this paper was to develop and test methods to detect and mitigate ransomware attacks. State of the art detection methods made by others are presented along with their own conclusion. We further evaluate whether these could work for detecting ransomware in the future. This is done by analyzing if there exists vulnerabilities that ransomware can exploit and use in future versions to avoid detection and mitigation. Several methods and the underlying theory are proposed as possible methods for detection, whereof two of these are implemented and tested.

We wish we had more time to test more detection methods and test those implemented more throughout. For the testing environment, we looked at publicly available solutions, however, none of them fit our criteria. We therefore spend a significant amount of time, developing our own secure and reliable testing environment, utilizing virtual machines, a central server and segregated development and test networks.

Originally the paper also wanted to address how to mitigate ransomware attacks, however, research showed, that mitigation of ransomware is relatively trivial, and therefore does not require much testing, only development time and was therefore not prioritized.

Our detection method, showed that detecting ransomwares using honeypots is a very feasible strategy. Our proof-of-concept implementation had a successrate of 77%, and we are confident that with further development this would be higher. Using Game Theory to analyze the optimal distribution of honeypots on the system we found that the optimal strategy is random placement in every folder, with lots of file types for future ransomware.

Although, in theory, using shannon entropy would be a better option than honeypots in the detection, the tests showed it to perform significantly worse than honeypots. Several flaws in the implementation caused this. When analyzing the detection methods with focus on false positives then the honeypot solution is much more reliable.

Chapter 9

Future Works

In this chapter the future of ransomware and this project will be discussed. First, examples upon additional work that could be made for this project is given, this also includes work that was originally outside of the scope of this thesis. Next improvements upon the testing environment is made. Finally an estimate upon how ransomware and its counters will be in the future.

9.1 Robustness

Since robustness never were in the scope of this thesis it naturally needs to be improved. The detection method of this work is started from an executable file and is dependant upon a DLL file in order to post results. This process can easily be terminated by a ransomware. A ransomware scanning active processes upon a system shutting down processes that do not have a crucial role for the system, would be able to find and stop the detection method before it even began encrypting files.

To accommodate this, the program needs to be implemented upon a lower level of the machine just like antivirus is. This is done by hooking the process into the kernel and prevent other sources from deleting or stopping this process. By having the detection method secured from stopping by a malicious program the ransomware now needs to avoid detection or counter the mitigation methods.

9.2 Mitigation

The current program stops the ransomware by killing the process responsible for encrypting the files. This has proven to be effective against 77% of the ransomwares tested against, but if there is another process for the ransomware, monitoring the encryption process, that process could easily start another encryption process. Therefore the parents for the found process needs to be terminated as well, this goes for all parents and children of the ransomware tree such that no process is left. After all the processes have been terminated the next thing to remove is the registry changes and files placed by these processes. Since ransomwares need persistence they often place files in several directories. To find these files a log of every process activity is needed. This log must contain every action ever made by the processes just terminated and store information going back several months in order to ensure complete removal of the ransomware.

Even though the ransomware has been removed completely it might still have encrypted some files on the system before its termination. In order to restore these files a backup is needed. This backup can either be made by the user and stored elsewhere, or it can be stored locally using VSS. As explained earlier in this thesis the vssadmin has flaws that might be abused by malicious programs.

Preferably the detection method is effective enough that the damage done to the system is in such a scale that it can be considered insignificant. Optimally the detection method is so effective that a ransomware is detected before it encrypts the first file.

The method used to find the process responsible for encrypting the files is using a third party program. This significantly increases the time from the detection is made till the process is shut down. To improve this an integrated method is needed. This method can, like procmon, monitor the different processes that views, changes or does any action to a single file. The method also needs to be able to return a process id of a given process. If such a method was implemented the time from detection to mitigation of a ransomware would be noticeably lower.

9.3 Detection methods

The methods implemented in this thesis are not perfectly attuned as detection methods and act more like proof of concept. Continuous testing of the detection methods would improve these to faster detection of ransomware.

This means that the shannon entropy would be finely balanced in order to detect signs of ransomware most efficiently while keeping the false positives ratio as low as possible.

For honeypots this means that the honeypots could be more strategically placed instead of the random placement they currently hold. By placing honeypots according to the possible orders of encryption the detection rate would be much faster. If only honeypots are encrypted during a ransomware attack then no harm has come to the actual system.

In order to lower the chance of false positives significantly for honey pots, the honeypots should be hidden from the user and given read only access such that the user will have a much lower chance of accessing and changing the honeypots by accident.

The detection methods made are currently using filemon to detect changes and modifications in files. It is yet to be tested whether that method of detection can be avoided by processes.

A special developed ransomware might be able to avoid detection from a single type of detection method or maybe even two. But different kinds of detection methods combined might increase the difficulty for ransomwares to stay hidden.

Several detection methods that help each other in detecting a ransomware could have a much higher percentage of ransomware detection than just a single one. Therefore we recommend that a tiered solution with combination of several types of detection methods is made, in order to preemptively counter future ransomwares.

9.4 Testing environment

One of the major problem with this thesis' testing environment was when testing new code for the systems implemented onto the virtual machines. The process of adding the code, take the snapshots, proper naming, restoring the correct snapshot and then run the test took several minutes in order to test code implementations upon the virtual machines. This could be improved for future works by having a test environment where instead of manually adding the code tested onto the system and going through the process of managing snapshots, the testing environment would instead automatically deploy the testing software, such that the newest software would be downloaded and tested upon the virtual machine. By doing so, one would save a large amount of time upon testing if code is correctly implemented and working as intended when tested with a live ransomware, while it also reduces the heavy use of hard disk space for snapshots.

9.5 Future challenges

The concept of ransomware has been around for a long time, but only in the last couple of years has it been a threat to every person connected to the internet. Despite this, new ransomware is created at an alarmingly fast rate and the targets for ransomware has already spread to more complex devices such as smartwatches and smartphones. Even though these ransomwares were locker ransomware and not crypto, it still shows that the target group for ransomware is expanding. A few potential examples for ransomware targets are:

Internet of things

More and more physical devices are connected to the internet. These devices were previously deemed irrelevant to have an internet connection too, but since smarthouses became more popular the number of devices connected to the internet has risen as well. These devices can be anything from fridges to the lighting and temperature of the house, all of these devices can be accessed and controlled from a mobile device the user has preferred. This means that if a malicious program were to gain access to these devices the program could set the temperature of the place to be very high or very low, it might be able to lock the doors if they are something that can be controlled as well. The possibilities are many and as of right now, the internet of things is a section of the internet that has not focused upon security.

Self driving cars

While it has already been proved that newer cars can be hacked, it is still not that large a market for malware yet. Hacking into a self driving car and gain full control of all systems in it could be a potential threat from ransomware. Imagine sitting in your car that is driving on the freeway, suddenly the doors lock and a note about paying a ransom pops up upon the display, if not payed within five minutes the car will accelerate and crash. Suddenly the thing of value held by a ransomware is not files or pictures of loved ones, it is life. Not only would every person most likely pay this ransom, making it extremely efficient, it would also damage the car industry tremendously. Like ransomware today that sometimes show the victim it has the capabilities to decrypt the files, a ransomware in a car might do the same by suddenly accelerating or testing the brakes in order to let the victim know that it really can steer the car into a wall.

To take control of a vehicle can also be used against the agricultural section that already have autonomous tractors and harvesters in use. Since it is very important in the farming industry to use the machines around the clock once the time comes, a delay in harvesting could be very expensive for a farmer since the crops might not be suited for harvesting later or the weather is not right anymore.

Medical equipment and applications
It is already commonly known that several hospitals and medical facilities has been hit by large scale attacks, the reason why ransomware prefers to target hospitals is due to the high probability of payout. When a hospital is infected with ransomware, not only does it cost a lot in downtime for the hospital, but a hospital is a place where time is not only money, but also life. As written above, when the value held is not something monetary, but life, then the ransom will be payed almost every time. This effect can be abused even more than it already is, by locking medical and surgical equipment during the time it is in use it might put the life of a person in the hands of the ransomware attacking.

Appendix A

Test results

This appendix contains the results of the tests made that are not shown in the report. It also shows the analysis of the different ransomwares that the detection methods has been tested against.

The unprocessed data can be found online on Github here.

A.1 Ransomware analysis

The following tables of data show the information of the ransomwares used in the tests.

A.1.1 Shortened hashvalues

To make things easier the hashvalues of the ransomwares has been reduced to simpler codenames.

sha1 hash	**Alias**
bbc5f026b644405522c9b0ca1b0d03f3d67779e6	R01
87420a2791d18dad3f18be436045280a4cc16fc4	R02
801e20dce982d4a60b26c9540f0e59bd6827b788	R03
d732a95bca679a310f45e02ef2bd192f4773787a	R04
f8dc6e615e3a9ba9a4c16d9f2dcb10fb16c9517f	R05
a95adf1580e78ae89759c16d9dd8c7dd8b169524	R06
ab67cc396889fa2be3d5122e409b099d9b70664f	R07
db5ee09153fc4a8ce2619db39e23ca56885f05e8	R08
60c1c6925ff2c7c49b40db3f0624a4b066a9dce5	R09

e654d39cd13414b5151e8cf0d8f5b166dddd45cb	R10
3a0b855dd052b2cdc6453f6cbdb858c7b55762b0	R11
35719ee58a5771156bc956bcf1b5c54ac3391593	R12
3d8039ba03a056fa455f8764f8ad8f59325144c7	R13
7dace304baf7800fb2bde81efcfbfeca374fb836	R14
c2500c9587a4f68df63b953aff9e4ffc446ece18	R15
989493c9fb948792458ed00b16c2dd57836b7b66	R16
01158e7529b21878460285a6dac6d0d1979045e2	R17
1bfb94b73856fe5611e615e078ddba444ff769f5	R18
9434a9b4f3e17a66de0ca3f7c1fd4d5e88ddc188	R19
bc5b55f5e4a2e8f32b82b7b21bc8c46aecd15384	R20
4fc7a663df94602906aa15a36cd6a3b257fe30d4	R21
4521e56c70bac58ca750791d1820caeb21496717	R22
91e8c5356defbb129dfd694e12fae72b30ff0f8c	R23
0087c0edc0dd8f154880abf57f156751922eb771	R24
c2f9927d5f5a8b50f18fa0a91684c9de02345201	R25
28f52b2a598428304b0a9032883c41c0817a58f5	R26
56cdf5a8d3187a534b70376c45a7fd17a71f9164	R27
96a466c5dbc7e4a10257afb3bd8b790f965084d9	R28
054f9db834e37459f10b83f56691a5d6e7f28334	R29
c656658c2ca2dbbd6f24e4b4ae801218ee828936	R30
2c8af799af11e03abc5face54f3943c2b3071203	R31
3c2d77985495edc6cef1f69d4ee6d6224119e4a2	R32
40c50cf9c1849c580eca133eca7d7d13436fbe35	R33
7a854db1ad7a94f356cf091ae2db4c0d4cb6b8a7	R34
ba6c5915598d45089f558ce10271eb729e168ad8	R35
0d8bb8222f1a324e048fb293011db5621ea8299c	R36
c2f278a572d0f00b51bdb5645de5afa5945b17df	R37
6f13afa7252b184098ba8b8a23bcb070a4cf326c	R38
a3148733ece4263949a921803c309ccf96d57496	R39
b7f9dd8ee3434b35fbb3395f69ff43fd5112a0c6	R40
aac5c1a4d9af47b9695954ab61c910804343a808	R41
bd4cbbfccf4f47656f767ebe473b6d225cc5865e	R42
628610489c41e78617f4e51d0d0143a07b245f85	R43
383a448b39b3eb8917cf36661996ca2c933ae53e	R44
e4ff07aed054f6bb044464fa151ceb9f76711fce	R45
a91d0e481699281efab888356ee718f6669659ab	R46
09367487551302a68e35b57757ae0bdf27227e01	R47
ed5407f8c89976172b67d68ac7bd7c55c2917068	R48
8285db1c3d05bbacc18e6851f6163732d9c87f84	R49
be85a2e4a9283044d7bd99c3bb90fe58003042ef	R50

8a45eb782761d683e5af7b0146da3c5fd6a8d473	R51
95deb2721b418f05a0b6a4cb4fa94c8c52f2fb73	R52
6fd0fe811ea54f139dc68202f52ebf969c2a5fff	R53
3bf6e6af23d8a452ad64a11423c8da5119aac671	R54
dbacf6039d6c8c8c3adc4bf298b5ee2d28938b2f	R55
6aef7d5a462268c438c8417ee0da3f130b8aa84a	R56
ab40f96fd8709315373cf390d0d9954613e55b2d	R57
dde70ab8312fcc9bb90bc45ac5ae13484f4bc45d	R58
27ec595e01e4c89fb17a895bced8b84871355df4	R59
8ee56c28b8e581e4a096e2ff81f6eb28f673c8b4	R60
6e534dc2c2d6894db95d796b958d6f6d49f9ce41	R61
f7e1a3e4d976253c903eef486c50336e8a8c7c4c	R62
3b6762efe73183bd93420f0109294a419c835d86	R63
47d2c5a68e96ae7bc43f305a7d5df082f93c623e	R64
ed92d1cf00da6a44316aedc6e872252cd72b1c17	R65

A.1.2 Ransomware properties

This table shows whether the ransomware deletes VSS, when it was submittet to reverse.it, what type of encryption pattern it uses and what ransomware it is assumed to be. The full description for the encryption pattern can be found in 7.2

Alias	VSS	Submittet	Encryption	Assumed ransomware
R01	Yes	-	1	Fantom
R02	Yes	2017-05-12	1	WannaCry
R03	Yes	2017-05-05	1	Razy
R04	Yes	2017-03-22	1	Symmi
R05	Yes	2017-04-28	2	Unknown
R06	Yes	2016-11-25	2	Cerber
R07	No	2017-05-02	2	Unknown
R08	Yes	2017-05-15	3	Zusy
R09	Yes	2017-05-14	3	Unknown
R10	Yes	2015-02-26	2	Zusy / TeslaCrypt
R11	No	2016-01-09	2	Zusy / Vipasana
R12	No	2016-01-09	2	Zusy / Vipasana
R13	No	2013-06-04	2	Xorist / Kazy
R14	No	-	2	Kazy / Xorist
R15	No	2014-03-23	4	Xorist
R16	No	2012-04-09	4	Barys
R17	Yes	2016-03-03	2	TeslaCrypt / Midie
R18	-	-	-	-
R19	No	2013-06-22	4	Xorist
R20	No	2014-11-30	2	Unknown
R21	No	2014-02-20	4	Xorist
R22	No	2014-07-28	4	Xorist
R23	Yes	2016-01-04	2	Alphacrypt
R24	No	2012-12-01	4	Dropper
R25	Yes	2016-02-24	2	TeslaCrypt
R26	Yes	2015-09-03	2	TeslaCrypt / Symmi
R27	No	2012-08-07	4	Xorist
R28	Yes	2016-02-02	2	Deshacop / TeslaCrypt
R29	No	2013-03-22	4	Xorist
R30	No	2011-03-02	2	Xorist / Kazy
R31	Yes	2016-03-06	2	TeslaCrypt
R32	Yes	2015-08-25	2	Deshacop / TeslaCrypt
R33	No	2011-05-31	2	Xorist / Symmi
R34	Yes	2015-08-30	2	Cripack / TeslaCrypt

R35	Yes	2015-08-21	2	Bitman / TeslaCrypt
R36	No	-	4	-
R37	Yes	2015-08-06	2	Deshacop / Cryptowall 3.0
R38	No	2013-06-19	4	Xorist / Kazy
R39	No	2016-05-04	2	CryptoLocker
R40	Yes	2016-02-03	1	HydraCrypt
R41	No	2011-09-21	2	Xorist
R42	Yes	2015-12-06	2	TeslaCrypt
R43	Yes	2016-02-26	2	TeslaCrypt
R44	Yes	2016-03-13	2	TeslaCrypt
R45	Yes	2015-12-03	2	Cripack / TeslaCrypt
R46	No	2013-01-17	2	Xorist / Kazy
R47	No	2013-06-25	2	Usteal
R48	No	2012-09-09	2	Xorist
R49	No	2012-06-21	2	Xorist
R50	No	2013-03-25	2	Xorist
R51	Yes	2015-08-22	2	Deshacop / TeslaCrypt
R52	No	2013-05-08	2	Xorist / Kazy
R53	Yes	2016-03-10	2	TeslaCrypt
R54	No	2015-04-24	2	CryptoLocker
R55	Yes	2015-08-25	2	Deshacop / TeslaCrypt
R56	No	2015-02-03	5	Androm
R57	Yes	2015-09-02	2	TesCrypt
R58	No	2013-07-27	2	Xorist
R59	Yes	2015-08-29	-	Cripack / TeslaCrypt
R60	No	2013-06-18	2	Xorist
R61	No	2014-04-09	2	Graftor
R62	Yes	2015-04-29	2	Symmi / TeslaCrypt
R63	No	2016-05-06	2	Zygug / Xorist
R64	No	2012-02-22	2	Heur
R65	Yes	2015-04-28	2	CryptoLocker

A.1.3 Ransomware encrypted filetypes

This table shows the different filetypes that each ransomware has encrypted in the tests made.

R01 7z, anb, bak, bmp, c, cpp, csv, dist, doc, docx, dump, e01, exe, fantom, gif, gitignore, gz, h, hhconfig, hhi, jar, java, jpg, json, lock, log1, log2, m, m4, md, mdxml, mp4, mw, nc, odp, ova, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, pml, png, pot, ppt, pptx, r, rar, red, sha256, sql, swf, tex, tif, txt, url, w32, wav, wmv, xls, xlsx, xml, yml, zip

R02 7z, bak, bat, bmp, c, cpp, csv, dll, doc, docx, eky, exe, gif, gz, h, jar, java, jpg, lnk, log1, log2, mp4, odp, pdf, php, pky, pl, png, pot, ppt, pptx, rar, res, sql, swf, tif, tmp, txt, vbs, wav, wmv, xls, xlsx, zip

R03 7z, anb, bak, bmp, c, cpp, csv, dist, doc, docx, dump, e01, exe, gif, gitignore, granit, h, hhconfig, hhi, jar, java, jpg, json, lock, log1, log2, m4, md, mdxml, mp4, mw, odp, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, pml, png, pot, ppt, pptx, r, rar, red, rst, sha256, sql, swf, tex, tif, txt, url, w32, wav, wmv, xls, xlsx, xml, yml, zip

R04 7z, anb, bak, bmp, c, cpp, csv, dist, doc, docx, dump, exe, gif, gitignore, gz, h, hhconfig, hhi, ini, jar, java, jpg, json, lnk, lock, log1, log2, m4, md, mdxml, mw, odp, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, png, pot, ppt, pptx, r, rar, red, rst, sha256, sql, swf, tex, tif, txt, url, w32, xls, xlsx, xml, yml, zip

R05 7z, anb, bak, bmp, c, cpp, csv, dist, doc, docx, dump, exe, gif, gitignore, gz, h, hhconfig, hhi, ini, jar, java, jpg, json, lnk, lock, log1, log2, m4, md, mdxml, mp4, mw, odp, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, png, pot, ppt, pptx, r, rar, red, rst, sha256, sql, swf, tex, tif, txt, url, w32, xls, xlsx, xml, yml, zip

R06 7z, ad4f, bak, bmp, c, cpp, csv, doc, docx, exe, gif, h, hta, jar, java, jpg, json, lock, log1, log2, md, mp4, odp, pdf, php, pl, pml, png, pot, ppt, pptx, rar, sql, swf, tex, tif, txt, wav, wmv, xls, xlsx, xml, zip

R07 7z, bak, bmp, doc, docx, exe, gz, html, jpg, log1, log2, m, mp4, pdf, pec, ppt, pptx, rar, sql, tif, txt, xls, xlsx, zip

R08 7z, anb, bak, bmp, c, cpp, csv, dist, doc, docx, dump, e01, exe, gif, gitignore, gz, h, hhconfig, hhi, html, ini, jar, java, jpg, json, lnk, lock, log1, log2, m4, md, mdxml, mp4, mw, nc, odp, ova, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, pml, png, pot, ppt, pptx, r, rar, red, redproject, rst, search-ms, searchconnector-ms, sha256, sql, swf, tex, tif, txt, url, w32, wallet, wav, wmv, xls, xlsx, xml, yml, zip

R09 7z, anb, bak, bmp, c, cpp, crypt, csv, dist, doc, docx, dump, e01, exe, gif, gitignore, gz, h, hhconfig, hhi, html, ini, jar, java, jpg, json, lnk, lock, log1, log2, m4, md, mdxml, mp4, mw, nc, odp, ova, pcap, pcapng, pdf, php, phpb, phpt, pl, pli, pml, png, pot, ppt, pptx, r, rar, red, redproject, rst, search-ms, searchconnector-ms, sha256, sql, swf, tex, tif, txt, url, w32, wav, wmv, xls, xlsx, xml, yml, zip

R10 7z, bmp, csv, doc, docx, ecc, exe, ini, jpg, json, lnk, log1, log2, odp, pcap, pdf, png, ppt, pptx, rar, txt, wmv, xls, xlsx, zip

R11 7z, cbf, csv, doc, docx, etl, exe, ini, jpg, log1, log2, odp, pdf, ppt, rar, txt, xls, xlsx, xml, zip

R12 7z, bak, cbf, csv, doc, docx, etl, exe, ini, jpg, log1, log2, odp, pdf, ppt, rar, txt, xls, xlsx, xml, zip

R13 7z, bmp, doc, docx, etl, exe, gif, ini, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R14 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, telka, txt, wav, wmv, xls, xlsx, zip

R15 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R16 7z, bmp, doc, docx, exe, gif, jpg, lnk, lock, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R17 7z, csv, doc, docx, exe, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, txt, wmv, xls, xlsx, zip

R18 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zalk, zip

R19

R20 bmp, doc, docx, exe, jpg, lnk, log1, log2, md, pdf, ppt, pptx, rar, txt, xls, xlsx, zip

R21 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R22 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R23 7z, bmp, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, vvv, wmv, xls, xlsx, zip

R24 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R25 7z, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R26 7z, abc, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R27 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R28 7z, abc, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R29 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R30 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, php, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zalupa, zip

R31 7z, bak, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R32 7z, abc, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R33 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R34 7z, abc, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R35 7z, aaa, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R36 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R37 7z, aaa, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R38 5043, 7z, csv, doc, docx, exe, jpg, log1, log2, pdf, ppt, rar, txt, xls, xlsx, xml, zip

R39 7z, bmp, csv, doc, docx, exe, gif, jpg, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, tif, txt, wav, wmv, xls, xlsx, zip

R40 7z, bak, bmp, c, cpp, csv, doc, docx, exe, gif, gz, h, ini, java, jpg, log1, log2, m4, md, mp4, nc, odp, pdf, php, pl, png, pot, ppt, pptx, r, rar, sql, swf, tex, tif, txt, wmv, xls, xlsx, xml, zip

R41 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R42 7z, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, vvv, wmv, xls, xlsx, zip

R43 7z, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R44 7z, bak, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R45 7z, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, vvv, wmv, xls, xlsx, zip

R46 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R47 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R48 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R49 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R50 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R51 7z, aaa, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R52 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, txt, wav, wmv, xls, xlsx, zip

R53 7z, bak, csv, doc, docx, exe, htm, html, jpg, log1, log2, m, mp3, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R54 7z, bmp, csv, doc, docx, ecc, exe, jpg, json, lnk, log1, log2, m, mp4, odp, pcap, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R55 7z, abc, bmp, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R56 7z, bmp, c, cpp, doc, docx, exe, jpg, log1, log2, md, odp, pdf, php, pl, ppt, pptx, rar, sql, txt, vqobftg, xlsx, zip

R57 7z, abc, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R58

R59 7z, abc, csv, doc, docx, exe, html, jpg, log1, log2, m, mp4, odp, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R60 7z, bmp, doc, docx, exe, gif, jpg, lnk, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, strip4you, txt, wav, wmv, xls, xlsx, zip

R61 7z, bmp, csv, doc, docx, exe, gif, jpg, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, tif, txt, wav, wmv, xls, xlsx, zip

R62 7z, bmp, csv, doc, docx, ecc, exe, jpg, json, lnk, log1, log2, m, mp4, odp, pcap, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

R63 7z, bmp, csv, doc, docx, exe, gif, jpg, locked, log1, log2, md, mp4, pdf, png, ppt, pptx, rar, tif, txt, wav, wmv, xls, xlsx, zip

R64 7z, bak, bmp, csv, doc, docx, exe, jpg, log1, log2, md, mp4, odp, pdf, png, ppt, pptx, rar, rsa1024, txt, wmv, xls, xlsx, xml, zip

R65 7z, bmp, csv, doc, docx, exe, ezz, jpg, json, lnk, log1, log2, m, mp4, odp, pcap, pdf, png, ppt, pptx, rar, sql, txt, wmv, xls, xlsx, zip

A.1.4 Detection method successrate against ransomware

In this table green means that the ransomware was detected, red means that it was not and yellow means that no data was returned when this test was made.

	hp1	hp2	hp5	hp10	sh3	sh5	sh10	sh15
R01	1	1	1	1	0	1	0	0
R02	1	1	1	1	-	0	1	1
R03	1	1	1	1	1	1	1	0
R04	1	1	1	1	1	1	1	1
R05	1	1	-	1	1	1	1	1
R06	1	0	1	1	-	1	0	0
R07	1	1	1	1	1	1	1	1
R08	1	1	1	1	1	1	1	1
R09	1	1	1	1	1	1	1	1
R10	1	1	-	-	0	-	-	-
R11	1	1	1	1	1	1	0	0
R12	1	1	1	1	1	1	1	-
R13	1	1	1	1	1	1	1	-
R14	1	1	-	1	1	1	1	1
R15	1	1	1	1	0	1	0	0
R16	1	1	1	1	-	-	0	0
R17	-	1	0	-	0	-	-	-
R18	0	0	-	-	-	-	-	-
R19	1	1	1	1	1	1	0	0
R20	1	1	1	1	0	0	1	0
R21	1	1	1	1	0	1	0	0
R22	1	1	1	1	1	1	0	0
R23	0	1	-	-	-	-	-	-
R24	1	1	1	1	0	0	0	0
R25	-	-	-	-	-	0	-	-
R26	1	1	1	1	1	1	1	1
R27	1	1	1	1	0	1	0	0
R28	1	1	1	1	1	1	1	1
R29	1	1	1	1	1	0	0	0
R30	1	1	1	1	1	1	1	0
R31	0	-	-	-	-	-	-	-
R32	1	1	1	1	-	1	1	1
R33	1	1	1	1	1	1	-	1
R34	1	1	1	1	1	1	1	1
R35	1	1	1	1	1	1	1	1
R36	1	1	1	1	0	0	0	0

R37	0	0	-	-	-	-	-	-
R38	1	1	1	1	1	0	0	0
R39	1	1	1	1	0	1	0	0
R40	1	1	1	1	-	-	-	-
R41	1	1	1	1	0	1	1	1
R42	-	0	-	-	0	-	-	-
R43	0	-	-	-	-	-	-	-
R44	-	-	-	-	0	-	0	0
R45	-	-	-	-	-	-	-	-
R46	1	1	1	1	0	0	0	0
R47	1	1	1	1	0	0	0	0
R48	1	1	1	1	0	0	0	0
R49	1	1	1	1	0	0	0	0
R50	1	1	1	1	0	0	0	0
R51	1	1	1	1	0	1	1	1
R52	1	1	1	1	0	1	0	0
R53	-	-	0	-	-	-	0	-
R54	-	-	-	-	1	-	-	-
R55	1	1	1	1	1	1	1	1
R56	-	-	-	-	-	-	-	-
R57	-	1	1	1	0	1	1	1
R58	1	1	1	-	1	1	1	-
R59	1	1	1	1	0	1	1	1
R60	1	1	1	-	1	1	1	1
R61	1	1	1	1	-	1	0	0
R62	-	-	-	-	-	-	-	-
R63	1	1	1	1	0	1	0	0
R64	1	1	1	1	1	1	0	0
R65	-	-	-	-	-	0	-	0

Appendix B

Computer Specifications

This appendix contains the specifications for the different setups used for the testing environment.

B.1 Datacollection server

The datacollection server was running a different setup than every other physical machine.

H/W path	Device	Class	Description
		system	OptiPlex 9010
/0		bus	0HJG5K
/0/0		memory	64KiB BIOS
/0/5c		processor	Intel(R) Core(TM) i7-3770S CPU @ 3.10GHz
/0/5c/38		memory	256KiB L1 cache
/0/5c/39		memory	1MiB L2 cache
/0/5c/3a		memory	8MiB L3 cache
/0/3b		memory	8GiB System Memory
/0/3b/0		memory	4GiB DIMM DDR3 Synchronous 1600 MHz (0.6 ns)
/0/3b/1		memory	4GiB DIMM DDR3 Synchronous 1600 MHz (0.6 ns)
/0/100		bridge	Xeon E3-1200 v2/3rd Gen Core processor DRAM Controller
/0/100/2		display	Xeon E3-1200 v2/3rd Gen Core processor Graphics Controller
/0/100/14		bus	7 Series/C210 Series Chipset Family USB xHCI Host Controller
/0/100/14/0	usb3	bus	xHCI Host Controller
/0/100/14/0/4		bus	USB 2.0 Hub
/0/100/14/0/4/1		input	Fujitsu Mouse
/0/100/14/0/4/2		input	USB Keyboard
/0/100/14/1	usb4	bus	xHCI Host Controller
/0/100/16		communication	7 Series/C210 Series Chipset Family MEI Controller #1
/0/100/19	eno1	network	82579LM Gigabit Network Connection
/0/100/1a		bus	7 Series/C210 Series Chipset Family USB Enhanced Host Controller #2
/0/100/1a/1	usb1	bus	EHCI Host Controller
/0/100/1a/1/1		bus	Integrated Rate Matching Hub
/0/100/1b		multimedia	7 Series/C210 Series Chipset Family High Definition Audio Controller
/0/100/1d		bus	7 Series/C210 Series Chipset Family USB Enhanced Host Controller #1
/0/100/1d/1	usb2	bus	EHCI Host Controller
/0/100/1d/1/1		bus	Integrated Rate Matching Hub
/0/100/1f		bridge	Q77 Express Chipset LPC Controller
/0/100/1f.2		storage	SATA Controller [RAID mode]
/0/100/1f.3		bus	7 Series/C210 Series Chipset Family SMBus Controller
/0/1	scsi0	storage	
/0/1/0.0.0	/dev/sda	disk	128GB LITEONIT LCT-128
/0/1/0.0.0/1	/dev/sda1	volume	111GiB EXT4 volume
/0/1/0.0.0/2	/dev/sda2	volume	8077MiB Extended partition
/0/1/0.0.0/2/5	/dev/sda5	volume	8077MiB Linux swap / Solaris partition
/0/2	scsi1	storage	
/0/2/0.0.0	/dev/cdrom	disk	DVD+-RW SN-208BB

Figure B.1: Hardware of the datacollection server

B.2 Test computers

The following contains the specification for the computers which the test were made upon, both physical and virtual.

```
------------------
System Information
------------------
      Time of this report: 7/04/2017, 16:54:23
             Machine name: DESKTOP-ELE8QOF
               Machine Id: {750A20DC-F1E0-474C-BEAD-01BEC683DE1B}
         Operating System: Windows 10 Enterprise 64-bit (10.0, Build 14393) (14393.rs1_release_sec.170427-1353)
                 Language: English (Regional Setting: English)
      System Manufacturer: Dell Inc.
             System Model: OptiPlex 9010
                     BIOS: BIOS Date: 09/19/12 10:43:37 Ver: A08.00
                Processor: Intel(R) Core(TM) i7-3770S CPU @ 3.10GHz (8 CPUs), ~3.1GHz
                   Memory: 8192MB RAM
      Available OS Memory: 8078MB RAM
                Page File: 6396MB used, 2962MB available
              Windows Dir: C:\Windows
          DirectX Version: DirectX 12
      DX Setup Parameters: Not found
         User DPI Setting: Using System DPI
       System DPI Setting: 96 DPI (100 percent)
          DWM DPI Scaling: Disabled
                 Miracast: Not Available
Microsoft Graphics Hybrid: Not Supported
           DxDiag Version: 10.00.14393.0000 64bit Unicode

------------------------
Disk & DVD/CD-ROM Drives
------------------------
      Drive: C:
 Free Space: 22.0 GB
Total Space: 238.0 GB
File System: NTFS
      Model: Samsung SSD 850 EVO 250GB

      Drive: E:
      Model: TSSTcorp DVD+-RW SN-208BB
     Driver: c:\windows\system32\drivers\cdrom.sys, 10.00.14393.0000 (English), 7/16/2016 13:41:53, 173056 bytes
```

Figure B.2: Hardware of the two identical physical computer

```
------------------
System Information
------------------
      Time of this report: 7/04/2017, 16:44:59
             Machine name: DESKTOP-K0FITH4
               Machine Id: {12F4C931-BA6B-4538-9542-F4E1E218475A}
         Operating System: Windows 10 Pro 64-bit (10.0, Build 14393) (14393.rs1_release_inmarket.161102-0100)
                 Language: Danish (Regional Setting: Danish)
      System Manufacturer: Dell Inc.
             System Model: OptiPlex 980
                     BIOS: Phoenix ROM BIOS PLUS Version 1.10 A02
                Processor: Intel(R) Core(TM) i5 CPU         660  @ 3.33GHz (4 CPUs), ~3.3GHz
                   Memory: 12288MB RAM
      Available OS Memory: 12086MB RAM
                Page File: 8706MB used, 5811MB available
              Windows Dir: C:\Windows
          DirectX Version: DirectX 12
      DX Setup Parameters: Not found
         User DPI Setting: Using System DPI
       System DPI Setting: 96 DPI (100 percent)
          DWM DPI Scaling: Disabled
                 Miracast: Not Available
Microsoft Graphics Hybrid: Not Supported
           DxDiag Version: 10.00.14393.0000 64bit Unicode

------------------------
Disk & DVD/CD-ROM Drives
------------------------
      Drive: C:
 Free Space: 34.9 GB
Total Space: 238.0 GB
File System: NTFS
      Model: Samsung SSD 850 EVO 250GB ATA Device

      Drive: D:
      Model: TSSTcorp DVD+-RW TS-L633C ATA Device
     Driver: c:\windows\system32\drivers\cdrom.sys, 10.00.14393.0000 (Danish), 7/16/2016 13:41:53, 173056 bytes
```

Figure B.3: Hardware of the physical computer that was slightly different

Name	Publisher	Installed On	Size	Version
7-Zip 16.04 (x64)	Igor Pavlov	17/03/2017	4.75 MB	16.04
FileZilla Client 3.25.0	Tim Kosse	17/03/2017	23.1 MB	3.25.0
GitHub	GitHub, Inc.	20/04/2017		3.3.4.0
Google Chrome	Google, Inc.	15/03/2017	46.4 MB	58.0.3029.110
Java 8 Update 121	Oracle Corporation	15/03/2017	188 MB	8.0.1210.13
Java 8 Update 121 (64-bit)	Oracle Corporation	15/03/2017	216 MB	8.0.1210.13
Microsoft OneDrive	Microsoft Corporation	15/03/2017	84.8 MB	17.3.6798.0207
Microsoft System CLR Types for SQL Server 2016	Microsoft Corporation	06/04/2017	7.93 MB	13.0.1601.5
Microsoft System CLR Types for SQL Server 2016	Microsoft Corporation	06/04/2017	11.3 MB	13.0.1601.5
Microsoft Visual C++ 2017 Redistributable (x64) - 14.1...	Microsoft Corporation	06/04/2017	23.4 MB	14.10.25008.0
Microsoft Visual C++ 2017 Redistributable (x86) - 14.1...	Microsoft Corporation	06/04/2017	19.5 MB	14.10.25008.0
Microsoft Visual Studio 2017	Microsoft Corporation	06/04/2017	148 MB	1.9.30330.1
Notepad++ (32-bit x86)	Notepad++ Team	15/03/2017	6.49 MB	7.3.3
Oracle VM VirtualBox 5.1.18	Oracle Corporation	17/03/2017	257 MB	5.1.18
Realtek High Definition Audio Driver	Realtek Semiconductor Corp.	17/03/2017	5.08 MB	6.0.1.6070
TeamViewer 12	TeamViewer	15/05/2017	86.6 MB	12.0.77242
Windows SDK AddOn	Microsoft Corporation	06/04/2017	304 KB	10.1.0.0
Windows Software Development Kit - Windows 10.0....	Microsoft Corporation	06/04/2017	1.14 GB	10.1.15063.137
WinSCP 5.9.4	Martin Prikryl	15/03/2017	30.9 MB	5.9.4

Figure B.4: Software installed on the physical computer

Name	Program	Version	Publisher	Installed On
Microsoft Windows (7)				
Update for Microsoft Windows (KB3150513)	Microsoft Windows		Microsoft Corporation	09/06/2017
Security Update for Microsoft Windows (KB4019472)	Microsoft Windows		Microsoft Corporation	10/05/2017
Security Update for Adobe Flash Player	Microsoft Windows		Microsoft Corporation	10/05/2017
Update for Microsoft Windows (KB4013418)	Microsoft Windows		Microsoft Corporation	15/03/2017
Update for Microsoft Windows (KB3199986)	Microsoft Windows		Microsoft Corporation	21/11/2016
Update for Microsoft Windows (KB3194623)	Microsoft Windows		Microsoft Corporation	21/11/2016
Security Update for Adobe Flash Player	Microsoft Windows		Microsoft Corporation	21/11/2016
Unspecified (1)				
Update for (KB2504637)		1	Microsoft Corporation	06/04/2017

Figure B.5: Windows updates installed physical computer

```
------------------
System Information
------------------
      Time of this report: 2/05/2017, 10:17:53
             Machine name: DESKTOP-HSQEBPO
               Machine Id: {24E50089-A964-402A-8D3D-707F0FAFFA75}
         Operating System: Windows 10 Enterprise 64-bit (10.0, Build 14393) (14393.rs1_release_inmarket.170303-1614)
                 Language: Danish (Regional Setting: Danish)
      System Manufacturer: innotek GmbH
             System Model: VirtualBox
                     BIOS: Default System BIOS
                Processor: Intel(R) Core(TM) i7-3770S CPU @ 3.10GHz, ~3.1GHz
                   Memory: 2048MB RAM
      Available OS Memory: 2048MB RAM
                Page File: 1489MB used, 1197MB available
              Windows Dir: C:\Windows
          DirectX Version: DirectX 12
      DX Setup Parameters: Not found
         User DPI Setting: Using System DPI
       System DPI Setting: 96 DPI (100 percent)
          DWM DPI Scaling: Disabled
                 Miracast: Not Available
Microsoft Graphics Hybrid: Not Supported
           DxDiag Version: 10.00.14393.0000 64bit Unicode
------------------------
Disk & DVD/CD-ROM Drives
------------------------
      Drive: C:
 Free Space: 19.7 GB
Total Space: 50.7 GB
File System: NTFS
      Model: VBOX HARDDISK

      Drive: D:
      Model: VBOX CD-ROM
     Driver: c:\windows\system32\drivers\cdrom.sys, 10.00.14393.0000 (Danish), 7/16/2016 13:41:53, 173056 bytes
```

Figure B.6: Hardware of the virtual computer

Name	Publisher	Installed On	Size	Version
7-Zip 16.04 (x64)	Igor Pavlov	06-04-2017	4,75 MB	16.04
Adobe Shockwave Player 12.2	Adobe Systems, Inc.	06-04-2017	33,9 MB	12.2.8.198
Dropbox	Dropbox, Inc.	15-06-2017	192 MB	28.4.14
Foxit Reader	Foxit Software Inc.	06-04-2017	196 MB	8.2.1.6871
Google Chrome	Google, Inc.	06-04-2017	46,4 MB	58.0.3029.110
Google Drive	Google, Inc.	06-04-2017	69,2 MB	2.34.5075.1619
Java 8 Update 121	Oracle Corporation	06-04-2017	188 MB	8.0.1210.13
Java 8 Update 121 (64-bit)	Oracle Corporation	06-04-2017	216 MB	8.0.1210.13
LibreOffice 5.3.2.2	The Document Foundation	06-04-2017	800 MB	5.3.2.2
Microsoft OneDrive	Microsoft Corporation	10-05-2017	84,8 MB	17.3.6799.0327
Microsoft Silverlight	Microsoft Corporation	06-04-2017	101 MB	5.1.50905.0
Microsoft Visual C++ 2015 Redistributable (x86) - 14.0...	Microsoft Corporation	06-04-2017	19,5 MB	14.0.24215.1
Mozilla Firefox 52.0.2 (x86 en-GB)	Mozilla	10-05-2017	181 MB	52.0.2
Mozilla Maintenance Service	Mozilla	06-04-2017	256 KB	52.0.2
Oracle VM VirtualBox Guest Additions 5.1.18	Oracle Corporation	06-04-2017		5.1.18.0
Skype™ 7.34	Skype Technologies S.A.	06-04-2017	171 MB	7.34.103
Spotify	Spotify AB	06-04-2017		1.0.52.725.g943b26a8
VLC media player	VideoLAN	06-04-2017	128 MB	2.2.4

Figure B.7: Software installed on the virtual computer

Name	Program	Version	Publisher	Installed On
Microsoft Silverlight (1)				
Microsoft Silverlight 5.1.50905.0	Microsoft Silverlight	5.1.50905.0	Microsoft Corporation	06-04-2017
Microsoft Windows (6)				
Update for Microsoft Windows (KB4015438)	Microsoft Windows		Microsoft Corporation	05-04-2017
Security Update for Adobe Flash Player	Microsoft Windows		Microsoft Corporation	17-03-2017
Update for Microsoft Windows (KB4013418)	Microsoft Windows		Microsoft Corporation	17-03-2017
Update for Microsoft Windows (KB3199986)	Microsoft Windows		Microsoft Corporation	21-11-2016
Update for Microsoft Windows (KB3194623)	Microsoft Windows		Microsoft Corporation	21-11-2016
Security Update for Adobe Flash Player	Microsoft Windows		Microsoft Corporation	21-11-2016

Figure B.8: Windows updates installed virtual computer

Appendix C

Database tables and structure

This appendix contains the SQL structures of the different test tables. The table for the different test methods are all similar to another.

```
mysql> show columns from quicktester;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| RansomwareName       | varchar(100) | NO  | PRI | NULL    |       |
| Fetched              | timestamp   | YES  |     | NULL    |       |
| Posted               | timestamp   | YES  |     | NULL    |       |
| FileChangedOnHash    | varchar(1)  | YES  |     | NULL    |       |
| FileChangedOnWatcher | varchar(1)  | YES  |     | NULL    |       |
| Active               | varchar(1)  | YES  |     | NULL    |       |
| TakenByBaseline      | varchar(45) | YES  |     | NULL    |       |
| TestedByBaseline     | varchar(45) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

Figure C.1: SQL structure of the quicktester table

```
mysql> show columns from baseline;
+-------------------------+--------------+------+-----+---------+-------+
| Field                   | Type         | Null | Key | Default | Extra |
+-------------------------+--------------+------+-----+---------+-------+
| RansomwareName          | varchar(100) | NO   | PRI | NULL    |       |
| Fetched                 | timestamp    | YES  |     | NULL    |       |
| Started                 | varchar(45)  | YES  |     | NULL    |       |
| Posted                  | timestamp    | YES  |     | NULL    |       |
| MonitorStatus           | varchar(45)  | YES  |     | NULL    |       |
| MonitorCount            | varchar(45)  | YES  |     | NULL    |       |
| CountChangedFiles       | varchar(45)  | YES  |     | NULL    |       |
| CountDeletedFiles       | varchar(45)  | YES  |     | NULL    |       |
| CountNewFiles           | varchar(45)  | YES  |     | NULL    |       |
| CountFilemonObservations | varchar(45) | YES  |     | NULL    |       |
| CPU                     | longtext     | YES  |     | NULL    |       |
| RAM                     | longtext     | YES  |     | NULL    |       |
| HDD                     | longtext     | YES  |     | NULL    |       |
| ThreadCount             | longtext     | YES  |     | NULL    |       |
| HandleCount             | longtext     | YES  |     | NULL    |       |
| ListChangedFiles        | longtext     | YES  |     | NULL    |       |
| ListDeletedFiles        | longtext     | YES  |     | NULL    |       |
| ListNewFiles            | longtext     | YES  |     | NULL    |       |
| ListFilemonObservations | longtext     | YES  |     | NULL    |       |
| TakenByHP1              | varchar(45)  | YES  |     | NULL    |       |
| TakenByHP2              | varchar(45)  | YES  |     | NULL    |       |
| TakenByHP5              | varchar(45)  | YES  |     | NULL    |       |
| TakenByHP10             | varchar(45)  | YES  |     | NULL    |       |
| TestedByHP1             | varchar(45)  | YES  |     | NULL    |       |
| TestedByHP2             | varchar(45)  | YES  |     | NULL    |       |
| TestedByHp5             | varchar(45)  | YES  |     | NULL    |       |
| TestedByHP10            | varchar(45)  | YES  |     | NULL    |       |
| TakenByShannon3         | varchar(45)  | YES  |     | NULL    |       |
| TakenByShannon5         | varchar(45)  | YES  |     | NULL    |       |
| TakenByShannon10        | varchar(45)  | YES  |     | NULL    |       |
| TakenByShannon15        | varchar(45)  | YES  |     | NULL    |       |
| TestedByShannon3        | varchar(45)  | YES  |     | NULL    |       |
| TestedByShannon5        | varchar(45)  | YES  |     | NULL    |       |
| TestedByShannon10       | varchar(45)  | YES  |     | NULL    |       |
| TestedByShannon15       | varchar(45)  | YES  |     | NULL    |       |
+-------------------------+--------------+------+-----+---------+-------+
35 rows in set (0.00 sec)
```

Figure C.2: SQL structure of the baseline table

```
mysql> show columns from hp1;
+---------------------------+---------------+------+-----+---------+-------+
| Field                     | Type          | Null | Key | Default | Extra |
+---------------------------+---------------+------+-----+---------+-------+
| RansomwareName            | varchar(100)  | NO   | PRI | NULL    |       |
| Fetched                   | timestamp     | YES  |     | NULL    |       |
| Started                   | varchar(45)   | YES  |     | NULL    |       |
| Posted                    | timestamp     | YES  |     | NULL    |       |
| MonitorStatus             | varchar(45)   | YES  |     | NULL    |       |
| MonitorCount              | varchar(45)   | YES  |     | NULL    |       |
| CountChangedFiles         | varchar(45)   | YES  |     | NULL    |       |
| CountDeletedFiles         | varchar(45)   | YES  |     | NULL    |       |
| CountNewFiles             | varchar(45)   | YES  |     | NULL    |       |
| CountFilemonObservations  | varchar(45)   | YES  |     | NULL    |       |
| CPU                       | longtext      | YES  |     | NULL    |       |
| RAM                       | longtext      | YES  |     | NULL    |       |
| HDD                       | longtext      | YES  |     | NULL    |       |
| ThreadCount               | longtext      | YES  |     | NULL    |       |
| HandleCount               | longtext      | YES  |     | NULL    |       |
| ListChangedFiles          | longtext      | YES  |     | NULL    |       |
| ListDeletedFiles          | longtext      | YES  |     | NULL    |       |
| ListNewFiles              | longtext      | YES  |     | NULL    |       |
| ListFilemonObservations   | longtext      | YES  |     | NULL    |       |
| NameOfShutdownRansomware  | longtext      | YES  |     | NULL    |       |
| Detected                  | varchar(45)   | YES  |     | NULL    |       |
| Shutdown                  | varchar(45)   | YES  |     | NULL    |       |
+---------------------------+---------------+------+-----+---------+-------+
22 rows in set (0.00 sec)
```

Figure C.3: SQL structure of the hp1 table

Appendix D

PHP Code

Courier

In this appendix, some of the code developed in PHP is attached.

The full original code can be found online on Github here.

D.1 Backend: DbHandler.php

The DbHandler is responsible for executing prepared statements to the MySQL database, using values received from the frontend. Below is a set of the primary methods used. It also uses a helper file, DbConnect which is basic file for communicating with the database.

Note that in the code below only the methods for HoneyPot 1 is shown, the methods for the other Honeypot tests and Shannon tests are the same, so they have been left out.

```
<?php

/**
 * Class to handle all db operations
 * This class has CRUD methods for database tables
 *
 **/
class DbHandler {

    private $conn;

    function __construct() {
        require_once dirname(__FILE__) .  /DbConnect.php ;
        // opening db connection
        $db = new DbConnect();
```

```
        $this->conn = $db->connect();
    }
//------------------------ Data extraction methods

    public function getDataBaseline($RansomwareName) {
        $stmt = $this->conn->prepare("SELECT␣*␣FROM␣baseline␣WHERE␣RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

    public function getDataHP1($RansomwareName) {
        $stmt = $this->conn->prepare("SELECT␣*␣FROM␣hp1␣WHERE␣RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }


//------------------------ Quick Tester methods

    public function getQuickRansomware() {
        $stmt = $this->conn->prepare("SELECT␣*␣FROM␣ quicktester␣WHERE␣Fetched␣is␣NULL␣limit␣1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

        public function postQuickFetched($RansomwareName){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE␣quicktester␣SET␣Fetched␣=␣CURRENT_TIMESTAMP␣WHERE␣RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
                // Check for successful insert/update
        if ($result) {
                        // Data successfully inserted/updated
            return  $result;
        } else {
                        // Failed to insert/update data
            return DATA_POST_FAILED;
        }
                return $response;
    }

        public function postQuickPosted($RansomwareName, $
```

```
            FileChangedOnHash, $FileChangedOnWatcher, $Active){
                $response = array();
        $stmt = $this->conn->prepare("UPDATE quicktester SET Posted = CURRENT_TIMESTAMP, FileChangedOnHash=?, FileChangedOnWatcher=?, Active=? WHERE RansomwareName=?");
        $stmt->bind_param("iiis", $FileChangedOnHash, $FileChangedOnWatcher, $Active, $RansomwareName);
        $result = $stmt->execute();
                $stmt->close();
                if ($result) {
            return  $result;
        } else {
        return DATA_POST_FAILED;
        }
        return $response;
    }

    public function getQuickHost() {
        $stmt = $this->conn->prepare("SELECT * FROM quicktester WHERE Fetched IS NOT NULL AND Posted IS NULL AND Active IS NULL limit 1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }


//----------------------- Baseline Tester methods

        public function getBaseRansomware() {
        $stmt = $this->conn->prepare("SELECT * FROM  quicktester WHERE Active=1 AND TakenByBaseline is NULL limit 1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

        public function postBaseFetched($RansomwareName){
        $response = array();

        $stmt = $this->conn->prepare("INSERT INTO baseline (RansomwareName, Fetched) VALUES (?, CURRENT_TIMESTAMP) ");
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
                        return  $result;
        } else {
            return DATA_POST_FAILED;
        }
```

```
        return $response;
    }

    public function postBasePosted($RansomwareName, $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE baseline SET Posted = CURRENT_TIMESTAMP, MonitorStatus=?, MonitorCount=?, CountChangedFiles=?, CountDeletedFiles=?, CountNewFiles=?, CountFilemonObservations=?, CPU=?, RAM=?, HDD=?, ThreadCount=?, HandleCount=?, ListChangedFiles=?, ListDeletedFiles=?, ListNewFiles=?, ListFilemonObservations=? WHERE RansomwareName=?");
        $stmt->bind_param("ssssssssssssssss", $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations, $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

    public function postBaseTaken($RansomwareName){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE quicktester SET TakenByBaseline = 1 WHERE RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

    public function postBaseTested($RansomwareName){
        $response = array();
                $stmt = $this->conn->prepare("UPDATE quicktester SET TestedByBaseline = 1 WHERE RansomwareName=?");
```

```
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

    public function getBaseHost() {
        $stmt = $this->conn->prepare("SELECT␣*␣FROM␣baseline␣WHERE␣Posted␣IS␣NULL␣ORDER␣BY␣Fetched␣DESC␣limit␣1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

        public function postBaseStarted($RansomwareName, $Started){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE␣baseline␣SET␣Started␣=?␣WHERE␣RansomwareName=?");
        $stmt->bind_param("ss", $Started, $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
                } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }


//------------------------ HoneyPot 1 PROCENT Tester methods

        public function getHP1Ransomware() {
        $stmt = $this->conn->prepare("SELECT␣*␣FROM␣ baseline ␣WHERE␣TakenByHP1␣is␣NULL␣limit␣1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

        public function postHP1Fetched($RansomwareName){
        $response = array();
        $stmt = $this->conn->prepare("INSERT␣INTO␣hp1␣(RansomwareName,␣Fetched)␣VALUES(?,␣CURRENT_TIMESTAMP)␣");
                $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
```

```php
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

        public function postHP1Posted($RansomwareName, $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations, $NameOfShutdownRansomware, $Detected, $Shutdown){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE hp1 SET Posted = CURRENT_TIMESTAMP, MonitorStatus=?, MonitorCount=?, CountChangedFiles=?, CountDeletedFiles=?, CountNewFiles=?, CountFilemonObservations=?, CPU=?, RAM=?, HDD=?, ThreadCount=?, HandleCount=?, ListChangedFiles=?, ListDeletedFiles=?, ListNewFiles=?, ListFilemonObservations=?, NameOfShutdownRansomware=?, Detected=?, Shutdown=?  WHERE RansomwareName=?");
        $stmt->bind_param("sssssssssssssssssss", $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations, $NameOfShutdownRansomware, $Detected, $Shutdown, $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

        public function postHP1Taken($RansomwareName){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE baseline SET TakenByHP1 = 1 WHERE RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
                if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
```

```php
        }
        return $response;
    }

        public function postHP1Tested($RansomwareName){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE baseline SET TestedByHP1 = 1 WHERE RansomwareName=?");
        $stmt->bind_param("s", $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
                if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }

    public function getHP1Host() {
        $stmt = $this->conn->prepare("SELECT * FROM hp1 WHERE Posted IS NULL ORDER BY Fetched DESC limit 1");
        $stmt->execute();
        $nextRansom = $stmt->get_result();
        $stmt->close();
        return $nextRansom;
    }

        public function postHP1Started($RansomwareName, $Started){
        $response = array();
        $stmt = $this->conn->prepare("UPDATE hp1 SET Started =? WHERE RansomwareName=?");
        $stmt->bind_param("ss", $Started, $RansomwareName);
        $result = $stmt->execute();
        $stmt->close();
        if ($result) {
            return  $result;
        } else {
            return DATA_POST_FAILED;
        }
        return $response;
    }
}
?>
```

D.2 Frontend: index.php

The index.php serves as the API interface, allowing for GET and POST messages. Every method has its own URI address, and takes in differents amounts of input and sends the relevant responses back. Below is a set of the primary methods.

```
<?php

require_once  ../include/DbHandler.php ;
require  .././libs/Slim/Slim.php ;

\Slim\Slim::registerAutoloader();

$app = new \Slim\Slim();

// User id from db - Global Variable
$user_id = NULL;


//--------------------- Data extraction

$app->get( /getdatabaseline , function() use ($app) {
                        $response = array();
            $db = new DbHandler();
            $RansomwareName = $app->request->params( RansomwareName );
            $result = $db->getDataBaseline($RansomwareName);
            $response["ransomware"] = array();
            // looping through result and preparing array
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
                $tmp["fecthed"] = $ransomware["Fetched"];
                $tmp["started"] = $ransomware["Started"];
                $tmp["posted"] = $ransomware["Posted"];
                $tmp["monitorStatus"] = $ransomware["MonitorStatus"];
                $tmp["monitorCount"] = $ransomware["MonitorCount"];
                $tmp["countChangedFiles"] = $ransomware["CountChangedFiles"];
                $tmp["countDeletedFiles"] = $ransomware["CountDeletedFiles"];
                $tmp["countNewFiles"] = $ransomware["CountNewFiles"];
                $tmp["countFilemonObservations"] = $ransomware["CountFilemonObservations"];
                $tmp["cpu"] = $ransomware["CPU"];
                $tmp["ram"] = $ransomware["RAM"];
                $tmp["hdd"] = $ransomware["HDD"];
                $tmp["threadCount"] = $ransomware["ThreadCount"];
                $tmp["handleCount"] = $ransomware["HandleCount"];
```

```
                $tmp["listChangedFiles"] = $ransomware["ListChangedFiles"];
                $tmp["listDeletedFiles"] = $ransomware["ListDeletedFiles"];
                $tmp["listNewFiles"] = $ransomware["ListNewFiles"];
                                $tmp["listFilemonObservations"] = $ransomware["ListFilemonObservations"];
              array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });

$app->get( /getdatahp1 , function() use ($app) {
            $response = array();
            $db = new DbHandler();
                        $RansomwareName = $app->request->params( RansomwareName );
            $result = $db->getDataHP1($RansomwareName);
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
                                $tmp = array();
                                $tmp["ransomware"] = $ransomware["RansomwareName"];
                                $tmp["fecthed"] = $ransomware["Fetched"];
                $tmp["started"] = $ransomware["Started"];
                $tmp["posted"] = $ransomware["Posted"];
                $tmp["monitorStatus"] = $ransomware["MonitorStatus"];
                $tmp["monitorCount"] = $ransomware["MonitorCount"];
                $tmp["countChangedFiles"] = $ransomware["CountChangedFiles"];
                $tmp["countDeletedFiles"] = $ransomware["CountDeletedFiles"];
                $tmp["countNewFiles"] = $ransomware["CountNewFiles"];
                $tmp["countFilemonObservations"] = $ransomware["CountFilemonObservations"];
                $tmp["cpu"] = $ransomware["CPU"];
                $tmp["ram"] = $ransomware["RAM"];
                $tmp["hdd"] = $ransomware["HDD"];
                $tmp["threadCount"] = $ransomware["ThreadCount"];
                $tmp["handleCount"] = $ransomware["HandleCount"];
                $tmp["listChangedFiles"] = $ransomware["ListChangedFiles"];
                $tmp["listDeletedFiles"] = $ransomware["ListDeletedFiles"];
                $tmp["listNewFiles"] = $ransomware["ListNewFiles"];
                $tmp["listFilemonObservations"] = $ransomware["ListFilemonObservations"];
                $tmp["nameOfShutdownRansomware"] = $ransomware["
```

```
                    NameOfShutdownRansomware"];
                $tmp["detected"] = $ransomware["Detected"];
                $tmp["shutdown"] = $ransomware["Shutdown"];
              array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });


//--------------------- QuickTester ransomware code

$app->get( /getquickransomware , function() {
            $response = array();
            $db = new DbHandler();
            $result = $db->getQuickRansomware();
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
              array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });

$app->post( /postquickfetched , function() use ($app) {
            // check for required params
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            // reading post params
            $RansomwareName = $app->request->params(
                RansomwareName );
            $db = new DbHandler();
            $res = $db->postQuickFetched($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data␣successfully␣
                    inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An␣error␣occured␣while␣
                    the␣insertion/update";
                        }
            echoRespnse(201, $response);
        });

$app->post( /postquickposted , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params(
                RansomwareName );
                        $FileChangedOnHash = $app->request->
                            params( FileChangedOnHash );
            $FileChangedOnWatcher = $app->request->params(
                FileChangedOnWatcher );
            $Active = $app->request->params( Active );
```

```
            $db = new DbHandler();
            $res = $db->postQuickPosted($RansomwareName, $FileChangedOnHash, $FileChangedOnWatcher, $Active);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                    }
            echoRespnse(201, $response);
        });

$app->get( /getquickhost , function() {
            $response = array();
            $db = new DbHandler();
            // fetching ransomwarename that the tester is working on
            $result = $db->getQuickHost();
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
              array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });


//--------------------- BaselineTester ransomware code

$app->get( /getbaseransomware , function() {
            $response = array();
            $db = new DbHandler();
            $result = $db->getBaseRansomware();
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
              array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });

$app->post( /postbasefetched , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postBaseFetched($RansomwareName);
            if ($res == 1 || $res == TRUE) {
```

```php
                $response["error"] = false;
                $response["message"] = "Data␣successfully␣inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An␣error␣occured␣while␣the␣insertion/update";
                        }
            echoRespnse(201, $response);
        });

$app->post( /postbaseposted , function() use ($app) {
            $response = array();
                        //parse json post
                        $json = $app->request->getBody();
                        $data = json_decode($json,true);
                                $RansomwareName = $data["RansomwareName"];
                                $MonitorStatus = $data["MonitorStatus"];
                                $MonitorCount = $data["MonitorCount"];
                                $CountChangedFiles = $data["CountChangedFiles"];
                                $CountDeletedFiles =  $data["CountDeletedFiles"];
                                $CountNewFiles = $data["CountNewFiles"];
                                $CountFilemonObservations = $data["CountFilemonObservations"];
                                $CPU = $data["CPU"];
                                $RAM = $data["RAM"];
                                $HDD = $data["HDD"];
                                $ThreadCount = $data["ThreadCount"];
                                $HandleCount = $data["HandleCount"];
                                $ListChangedFiles = $data["ListChangedFiles"];
                                $ListDeletedFiles = $data["ListDeletedFiles"];
                                $ListNewFiles = $data["ListNewFiles"];
                                $ListFilemonObservations = $data["ListFilemonObservations"];
                        $db = new DbHandler();
            $res = $db->postBasePosted($RansomwareName, $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
```

```
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                    }
            echoRespnse(201, $response);
        });

$app->post( /postbasetaken , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postBaseTaken($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                    }
            echoRespnse(201, $response);
        });

$app->post( /postbasetested , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postBaseTested($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                    }
            echoRespnse(201, $response);
        });

$app->get( /getbasehost , function() {
            $response = array();
            $db = new DbHandler();
            $result = $db->getBaseHost();
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
```

```
                $tmp["ransomware"] = $ransomware["RansomwareName"];
                array_push($response["ransomware"], $tmp);
            }
            echoRespnse(200, $response);
        });


$app->post( /postbasestarted , function() use ($app) {
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
                    $Started = $app->request->params( Started );
            $db = new DbHandler();
            $res = $db->postBaseStarted($RansomwareName, $Started);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data␣successfully␣inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An␣error␣occured␣while␣the␣insertion/update";
                        }
            echoRespnse(201, $response);
        });

//--------------------- HoneyPot 1 PROCENT ransomware code

$app->get( /gethp1ransomware , function() {
            $response = array();
            $db = new DbHandler();
            $result = $db->getHP1Ransomware();
            $response["ransomware"] = array();

            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
              array_push($response["ransomware"], $tmp);
            }

            echoRespnse(200, $response);
        });

$app->post( /posthp1fetched , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postHP1Fetched($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data␣successfully␣
```

```
                    inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                    }
            echoRespnse(201, $response);
        });

$app->post( /posthp1posted , function() use ($app) {
            $response = array();
                        $json = $app->request->getBody();
                        $data = json_decode($json,true);
                        $RansomwareName = $data["RansomwareName"];
                $MonitorStatus = $data["MonitorStatus"];
                $MonitorCount = $data["MonitorCount"];
                $CountChangedFiles = $data["CountChangedFiles"];
                $CountDeletedFiles =  $data["CountDeletedFiles"];
                $CountNewFiles = $data["CountNewFiles"];
                                $CountFilemonObservations = $data["CountFilemonObservations"];
                                $CPU = $data["CPU"];
                                $RAM = $data["RAM"];
                                $HDD = $data["HDD"];
                                $ThreadCount = $data["ThreadCount"];
                                $HandleCount = $data["HandleCount"];
                                $ListChangedFiles = $data["ListChangedFiles"];
                                $ListDeletedFiles = $data["ListDeletedFiles"];
                                $ListNewFiles = $data["ListNewFiles"];
                                $ListFilemonObservations = $data["ListFilemonObservations"];
                                $NameOfShutdownRansomware = $data["NameOfShutdownRansomware"];
                                $Detected = $data["Detected"];
                                $Shutdown = $data["Shutdown"];
            $db = new DbHandler();
            $res = $db->postHP1Posted($RansomwareName, $MonitorStatus, $MonitorCount, $CountChangedFiles, $CountDeletedFiles, $CountNewFiles, $CountFilemonObservations, $CPU, $RAM, $HDD, $ThreadCount, $HandleCount, $ListChangedFiles, $ListDeletedFiles, $ListNewFiles, $ListFilemonObservations, $NameOfShutdownRansomware, $Detected, $Shutdown);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
```

```
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                        }
            echoRespnse(201, $response);
        });

$app->post( /posthp1taken , function() use ($app) {
                        verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postHP1Taken($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                        }
            echoRespnse(201, $response);
        });

$app->post( /posthp1tested , function() use ($app) {
            verifyRequiredParams(array( RansomwareName ));
            $response = array();
            $RansomwareName = $app->request->params( RansomwareName );
            $db = new DbHandler();
            $res = $db->postHP1Tested($RansomwareName);
            if ($res == 1 || $res == TRUE) {
                $response["error"] = false;
                $response["message"] = "Data successfully inserted/updated";
            } else {
                $response["error"] = true;
                $response["message"] = "An error occured while the insertion/update";
                        }
            echoRespnse(201, $response);
        });

$app->get( /gethp1host , function() {
            $response = array();
            $db = new DbHandler();
            $result = $db->getHP1Host();
            $response["ransomware"] = array();
            while ($ransomware = $result->fetch_assoc()) {
              $tmp = array();
              $tmp["ransomware"] = $ransomware["RansomwareName"];
              array_push($response["ransomware"], $tmp);
```

```
                }
                echoRespnse(200, $response);
        });

$app->post( /posthp1started , function() use ($app) {
                        $response = array();
                        $RansomwareName = $app->request->params( RansomwareName );
                        $Started = $app->request->params( Started );
                $db = new DbHandler();
                $res = $db->postHP1Started($RansomwareName, $Started);
                if ($res == 1 || $res == TRUE) {
                    $response["error"] = false;
                    $response["message"] = "Data␣successfully␣inserted/updated";
                } else {
                    $response["error"] = true;
                    $response["message"] = "An␣error␣occured␣while␣the␣insertion/update";
                        }
                echoRespnse(201, $response);
        });

//--------------------- Helper methods

function echoRespnse($status_code, $response) {
    $app = \Slim\Slim::getInstance();
    // Http response code
    $app->status($status_code);

    // setting response content type to json
    $app->contentType( application/json );

    echo json_encode($response);
}

$app->run();
?>
```

Appendix E

C# Code

In this appendix, some of the code developed in C# is attached. Due to the full code is 236 separate files and 20594 lines of code, many of these repetition, only the most relevant code is represented.

The full original code can be found online on Github here.

E.1 Host controller

The host controller is executed upon the physical machine and is used to manage the virtual machine. The main part is two classes, one for controlling the system and one for handling the virtual machines.

E.1.1 Main control unit

```
 using  System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotHost
{
  class hostPocController
  {
    private static string nameOfMachine = "PocTester";
    private static string nameofStartUpSnapshot = "
    HoneyPot1POCsnapshotStartUp";
    private static string FULLRESPONSESTRING = "";
```

```
private static string NAMEONTEST = "Error";

private static readonly HttpClient client = new HttpClient();

//Every number here adds 5 seconds
static int thresholdForRuntime = 80 * 12;

//Hosts the baseline every 33 minute
public static void hostOfPOCTester()
{
  //Creates a virtualmachine controller
  VirtualMachineController tempVir = null;

  Boolean action = false;
  while (true)
  {
    //Instances a new virtual machine
    tempVir = new VirtualMachineController();

    //Starts up the machine
    tempVir.startVirtualMachine(nameOfMachine);

    Thread.Sleep(60000);

    getPocHP1Host();
    string temp = FULLRESPONSESTRING;

    Console.WriteLine(temp);

    int count = temp.Split( : ).Length - 1;

    action = false;

    int runs = 0;

    Console.WriteLine(temp);

    while (!action)
    {
      if (count > 1)
      {
        Console.WriteLine(temp);
        Console.WriteLine(count);
        getPocHP1Host();
        if (!temp.Equals(FULLRESPONSESTRING))
        {
          Console.WriteLine("Shutting down virtual machine due to post message");
          action = true;
        }
        runs++;
        Thread.Sleep(5000);

        if(runs >= thresholdForRuntime)
        {
```

```
                Console.WriteLine("Posting because no post has been made");
                action = true;
            }
        }
        else
        {
            Thread.Sleep(5000);
            getPocHP1Host();
            temp = FULLRESPONSESTRING;
            count = temp.Split ( : ).Length - 1;
        }
    }

    //Powers off the machine
    tempVir.poweroffVirtualMachine(nameOfMachine);
    Thread.Sleep(5000);

    //Restores the virtual machine to the original image
    tempVir.restoreVirtualMachine(nameOfMachine, nameofStartUpSnapshot);
    Thread.Sleep(10000);
  }
}

public static void getPocHP1Host()
{
  string responseString = "";
  try
  {
    client.DefaultRequestHeaders.Clear();
    client.DefaultRequestHeaders.ConnectionClose = true;
    responseString = client.GetStringAsync("http://192.168.8.102/v1/index.php/gethp1host").Result;

  }
  catch (Exception)
  {

    throw;
  }

  FULLRESPONSESTRING = responseString;
}

private static string findNAMEONTEST(string responsestring)
{
  int i = 0;
  int j = 0;
  foreach (char c in responsestring)
  {
    if (i == 5)
    {
      return responsestring.Substring(j, responsestring.Length - j - 4);
```

```
          }
          if (c.Equals( " ))
          {
            i++;
          }
          j++;
        }

        return␣"Could Not Find";
      }
    }
}
```

E.1.2 Virtual Machine Controller

```
 using  System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HoneyPotHost
{
  class VirtualMachineController
  {
    //Creates a process for the commandopromt
    private static Process cmd = new Process();

    //A function to power off the virtual machine
    public void poweroffVirtualMachine(string machineName)
    {
      cmd.StartInfo.FileName = "cmd.exe";
      cmd.StartInfo.RedirectStandardInput = true;
      cmd.StartInfo.RedirectStandardOutput = true;
      cmd.StartInfo.CreateNoWindow = true;
      cmd.StartInfo.UseShellExecute = false;
      cmd.Start();

      cmd.StandardInput.WriteLine(@"""C:\Program Files\Oracle\VirtualBox\VBoxManage.exe"" controlvm " + machineName + " poweroff");
      cmd.StandardInput.Flush();
      cmd.StandardInput.Close();
      cmd.WaitForExit();
    }

    //A function to restore the virtual machine
    public void restoreVirtualMachine(string machineName, string snapshotName)
    {
      cmd.StartInfo.FileName = "cmd.exe";
      cmd.StartInfo.RedirectStandardInput = true;
      cmd.StartInfo.RedirectStandardOutput = true;
      cmd.StartInfo.CreateNoWindow = true;
      cmd.StartInfo.UseShellExecute = false;
      cmd.Start();

      cmd.StandardInput.WriteLine(@"""C:\Program Files\Oracle\VirtualBox\VBoxManage.exe"" snapshot " + machineName + " restore " + snapshotName);
      cmd.StandardInput.Flush();
      cmd.StandardInput.Close();
      cmd.WaitForExit();
    }

    //A function to start the virtual machine
    public void startVirtualMachine(string machineName)
```

```
    {
      cmd.StartInfo.FileName = "cmd.exe";
      cmd.StartInfo.RedirectStandardInput = true;
      cmd.StartInfo.RedirectStandardOutput = true;
      cmd.StartInfo.CreateNoWindow = true;
      cmd.StartInfo.UseShellExecute = false;
      cmd.Start();

      cmd.StandardInput.WriteLine(@"""C:\Program Files\Oracle\VirtualBox\VBoxManage.exe"" startvm " + machineName);
      cmd.StandardInput.Flush();
      cmd.StandardInput.Close();
      cmd.WaitForExit();
    }
  }
}
```

E.2 Ransomware downloader

The following code is of the program designed to download the ransomware and run it upon the computer.

E.2.1 Main control unit

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotPOCRansomwareDownloader
{
  class Program
  {
    static void Main(string[] args)
    {
      ransomwareDownload();
    }

    public static void ransomwareDownload()
    {
      //Ensure that the ransomware will not be downloaded on the host computer
      if (Environment.MachineName.Contains("viruseater")) return;
      if (Environment.UserName.Contains("viruseater")) return;
      if (Environment.UserName.Contains("PoC-tester")) return;
      Thread.Sleep(2000);
      //The filepath is set to desktop
      serverCommunicator.setRansomwareFilePath();
      //Find the name of the ransomware
      serverCommunicator.getPOCHost();
      Thread.Sleep(100);
      Console.WriteLine(serverCommunicator.getNAMEONTEST());
      Thread.Sleep(100);

      //Download the ransomware
      serverCommunicator.downloadFileFTP();

      Thread.Sleep(100);

      //Inform the server that the ransomware is executed post this
      serverCommunicator.postPoCStarted();

      Thread.Sleep(100);

```

```
        //Execute the ransomware
        programExecuter.executeProgram(serverCommunicator.
      getRansomwareFilePath());

      }
    }
}
```

```
  using   System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HoneyPotPOCRansomwareDownloader
{
    class programExecuter
    {
      //Class to execute a program
      public static void executeProgram(string programPath)
      {
        Process.Start(programPath);
      }
    }
}
```

E.2.2 Server Communicator

```
  using   System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace HoneyPotPOCRansomwareDownloader
{
  class serverCommunicator
  {
    static string NAMEONTEST = "";
    static string RANSOMWAREFILEPATH = "";
    private static readonly HttpClient client = new HttpClient();

    public static void getPOCHost()
    {

      var responseString = client.GetStringAsync("http
    ://192.168.8.102/v1/index.php/gethp1host").Result;
      NAMEONTEST = findNAMEONTEST(responseString);
      Console.WriteLine(NAMEONTEST);

    }

    private static string findNAMEONTEST(string responsestring)
    {
      int i = 0;
      int j = 0;
      foreach (char c in responsestring)
      {
        if (i == 5)
        {
          return responsestring.Substring(j, responsestring.
    Length - j - 4);
        }
        if (c.Equals ( " ) )
        {
          i++;
        }
        j++;
      }

      return␣"what?";
    }

    public␣static␣void␣downloadFileFTP()
    {
      string␣ransomwareName␣=␣NAMEONTEST;

      string␣ftphost␣=␣"192.168.8.102";
```

```
      string␣ftpfilepath␣=␣"/VirusShare/"␣+␣ransomwareName;

      string␣ftpfullpath␣=␣"ftp://" + ftphost + ftpfilepath;

      using (WebClient request = new WebClient())
      {
        request.Credentials = new NetworkCredential("
    datacollector", "");
        byte[] fileData = request.DownloadData(ftpfullpath);

        using (FileStream file = File.Create(RANSOMWAREFILEPATH))
        {
          file.Write(fileData, 0, fileData.Length);
          file.Close();
        }
      }
    }

    public static async void postPoCStarted()
    {
      var values = new Dictionary<string, string>
      {
        {"RansomwareName", NAMEONTEST},
        {"Started", DateTime.Now.ToString("dd/MM/yyyy␣HH:mm:ss.
    fff") }
      };

      var content = new FormUrlEncodedContent(values);

      var response = client.PostAsync("http://192.168.8.102/v1/
    index.php/posthp1started", content).Result;

      var responseString = await response.Content.
    ReadAsByteArrayAsync();
    }



    public static string getNAMEONTEST()
    {
      return NAMEONTEST;
    }

    //Dynamic method of setting the path to the ransomware file
    public static void setRansomwareFilePath()
    {
      RANSOMWAREFILEPATH = Environment.GetFolderPath(Environment.
    SpecialFolder.Desktop) + "\\ransomware.exe";
    }

    public static string getRansomwareFilePath()
    {
      return RANSOMWAREFILEPATH;
    }
  }
```

```

}
```

E.3 Honeypot Prove of Concept

This appendix shows the detection method. Much of this code is reused in the other detection methods.

E.3.1 Main control unit

```
  using   HoneyPotPOC.PocLogger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotPOC
{
  class Program
  {
    //In addition, four paths needs to be set in PocLogger\Logger
    //static string PATH = @"C:\Users\PoC";
    static string PATH = @"C:\Users\PoC";
    static string BACKINGNAME = "backingFromProcMon";
    static string pathToBackingFile = @"C:\procmon\
    backingFileTest";
    static string ProcMonPath = @"C:\procmon\Procmon.exe";

    //Path to ransomware downloader
    static string RANSOMWAREDOWNLOADERPATH = @"C:\Software\
    HoneyPotPOCRansomwareDownloader\bin\Release\
    HoneyPotPOCRansomwareDownloader.exe";

    static void Main(string[] args)
    {
      //This wait is made such that a snapshot of the virtual
    machine could be made during the start of the program.
      Thread.Sleep(30000);
      honeyPotFileMonDetection();
    }

    public static void honeyPotFileMonDetection()
    {
      //Fetch the ransomwarename
      Logger.getPoCRansomware();

      Thread.Sleep(1000);
      //Inform the server that the ransomware has been fetched
      Logger.postPoCFetched();

      //Wait for response from the server
      while (!Logger.getHasFetched())
```

```
        {
          Thread.Sleep(500);
        }

        //Sets the correct values in different classes
        Logger.setRansomwareDownloaderPath(RANSOMWAREDOWNLOADERPATH
      );

        ActionTaker.setBackingName(BACKINGNAME);
        ActionTaker.setPathToBackingFile(pathToBackingFile);

        ProcMon.setPathToProcMon(ProcMonPath);
        BACKINGNAME = BACKINGNAME + 0;

        //Start the procmon
        var t = new Thread(() => ProcMon.createProcmonBackingFile(
      pathToBackingFile, BACKINGNAME));
        t.Start();

        Console.WriteLine(Logger.getNAMEONTEST());
        //Start the logger
        Logger.LogWriter(PATH);

        //Post that the ransomware succesfully has been tested
        Logger.postPoCTested();

        //Post the tested results
        Logger.postPoCPosted();


        Thread.Sleep(30000);
      }
    }
}
```

E.3.2 Filemon for honeypots

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Threading;
using System.Collections;

namespace HoneyPotPOC
{
  class FileMon
  {
    static int MONITORTIMEOUT = 60;
    static int thresholdNum = 1;
    public static int i = 0;
    public static int temp = 0;
    public static Dictionary<string, DateTime> eventNameAndTime =
     new Dictionary<string, DateTime>();
    private static Boolean hasMadeFirstDetection = false;
    private static DateTime firstDetectionTime = new DateTime();
    private static List<DateTime> threshold = new List<DateTime
    >();
    static Boolean stopLogging = false;

    //FileSystemWatcher can monitor changes in files
    private static FileSystemWatcher watcher = new
    FileSystemWatcher();

    public static void createFileWatcher(string path)
    {

      //The given path dictates what directory the watcher will
    monitor
      watcher.Path = path;

      //The NotifyFilters determine what the monitors triggers
    upon.
      //It can also be a change in size.
      watcher.NotifyFilter = NotifyFilters.Size | NotifyFilters.
    LastWrite | NotifyFilters.FileName;

      //The filter gives the watcher a specific filename to look
    for
      // "*honeypot.*" monitors every file with honeypot in the
    ending, and every format.
      watcher.Filter = "*honeypotbait*";

      //This tells the watcher when to react on different changes
      watcher.Created += new FileSystemEventHandler(OnChanged);
      watcher.Changed += new FileSystemEventHandler(OnChanged);
      watcher.Deleted += new FileSystemEventHandler(OnChanged);
      watcher.Renamed += new RenamedEventHandler(OnRenamed);
```

```

      watcher.EnableRaisingEvents = true;
      //IncludeSubdirectories does such that not only the directory given is monitored
      //but also every single subdirectory of the given directory
      watcher.IncludeSubdirectories = true;
    }


    //Event handler if an object is changed
    private static void OnChanged(object source, FileSystemEventArgs e)
    {
      Console.WriteLine("File: " + e.FullPath + " has been " + e.ChangeType);
      threshold.Add(DateTime.Now);
      List<DateTime> temp = new List<DateTime>();
      DateTime now = DateTime.Now;
      foreach (DateTime t in threshold)
      {
        if (60 < (now.Subtract(t).Seconds))
        {
          temp.Add(t);
        }
      }

      foreach (DateTime t in temp)
      {
        threshold.Remove(t);
      }

      //If threshold is reached, it makes a reaction
      if (threshold.Count > thresholdNum)
      {
        Console.WriteLine("Threshold reached. It s killing time");

        if (!hasMadeFirstDetection)
        {
          firstDetectionTime = DateTime.Now;
          hasMadeFirstDetection = true;
        }
        if (eventNameAndTime.ContainsKey(e.FullPath))
        {
          //Report it has been changed
          Console.WriteLine("File: " + e.FullPath + " has been " + e.ChangeType);
          if (MONITORTIMEOUT < (DateTime.Now.Subtract((DateTime)eventNameAndTime[e.FullPath])).TotalSeconds)
          {
            Console.WriteLine("Stopping the process");
            eventNameAndTime[e.FullPath] = DateTime.Now;
            ActionTaker.honeypotChange(e.FullPath);
          }
        }
```

```
        else
        {
          //Report it has been changed
          Console.WriteLine("File: " + e.FullPath + " has been "
    + e.ChangeType);
          eventNameAndTime.Add(e.FullPath, DateTime.Now);
          ActionTaker.honeypotChange(e.FullPath);
        }
      }
    }

    //Event handeler if an object is renamed
    private static void OnRenamed(object source, RenamedEventArgs
     e)
    {
      Console.WriteLine("Flie: {0} renamed to {1}", e.OldFullPath
    , e.FullPath);
    }

    public static DateTime getFirstDetected()
    {
      return firstDetectionTime;
    }

    public static void setWatcherToStop()
    {
      watcher.EnableRaisingEvents = false;
    }
  }
}
```

E.3.3 Procmon

```
  using   System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotPOC
{
  class ProcMon
  {
    private static Process cmd = new Process();
    private static string procMonPath = "";
    private static Boolean isHasherDone = false;

    //Starts procmon and gives a given path for the backing file
    public static void createProcmonBackingFile(string path, string backingName)
    {
      //Don t start procmon untill the hashing process is done
      while (!isHasherDone)
      {
        Thread.Sleep(500);
      }
      string backPath = path + @"\"␣+␣backingName;

      cmd.StartInfo.FileName␣=␣"cmd.exe";
      cmd.StartInfo.RedirectStandardInput␣=␣true;
      cmd.StartInfo.RedirectStandardOutput␣=␣true;
      cmd.StartInfo.CreateNoWindow␣=␣true;
      cmd.StartInfo.UseShellExecute␣=␣false;
      cmd.Start();

      cmd.StandardInput.WriteLine(@"start "␣+␣procMonPath␣+␣@" /quiet /minimized /backingfile "␣+␣path␣+␣"\\"␣+␣backingName␣+␣".PML");
      Console.WriteLine("Path to procMon file: "␣+␣path␣+␣"\\"␣+␣backingName);
      cmd.StandardInput.Flush();
    }

    //Shuts␣down␣procmon
    public␣static␣void␣procmonTerminator(string␣path,␣string␣backingName)
    {
      cmd.StartInfo.FileName␣=␣"cmd.exe";
      cmd.StartInfo.RedirectStandardInput␣=␣true;
      cmd.StartInfo.RedirectStandardOutput␣=␣true;
      cmd.StartInfo.CreateNoWindow␣=␣true;
      cmd.StartInfo.UseShellExecute␣=␣false;
      cmd.Start();
```

```
      cmd.StandardInput.WriteLine(procMonPath + " /waitforidle");
      cmd.StandardInput.WriteLine(procMonPath + " /terminate");
      Console.WriteLine("Path to procMon file: " + path + "\\" + backingName + ".PML");
      bool isProcMonTerminated = false;

      while (isProcMonTerminated == false)
      {
        Process[] pname = Process.GetProcessesByName("Procmon64");
        if (pname.Length == 0)
        {
          Console.WriteLine("Procmon is no longer running, continuing...");
          isProcMonTerminated = true;
        }
        else {
          Console.WriteLine("Procmon64 process is running!");
        }
        Thread.Sleep(50);
      }
    }

    //Lets procmon convert PML file to CSV
    public static void convertPMLfileToCSV(string path, string PMLfile, string CSVfile)
    {
      path = path + @"\";
      Process cmd = new Process();
      cmd.StartInfo.FileName = "cmd.exe";
      cmd.StartInfo.RedirectStandardInput = true;
      cmd.StartInfo.RedirectStandardOutput = true;
      cmd.StartInfo.CreateNoWindow = true;
      cmd.StartInfo.UseShellExecute = false;
      cmd.Start();

      cmd.StandardInput.WriteLine(@"start " + procMonPath + " /quiet /minimized /AcceptEula /SaveApplyFilter /saveas " + path + CSVfile + " /OpenLog " + path + PMLfile);
      Thread.Sleep(5000);
      int i = 0;
      long length = 0;
      while (!File.Exists(path + CSVfile))
      {
        try
        {
          length = new System.IO.FileInfo(path + CSVfile).Length;
        }
        catch (Exception)
        {
        }
        Thread.Sleep(50);
      }
      long temp = 0;
```

```
            while (length != temp)
            {
                i++;
                temp = length;
                Thread.Sleep(50);
                length = new System.IO.FileInfo(path + CSVfile).Length;
            }
            cmd.StandardInput.Flush();
            cmd.StandardInput.Close();
            cmd.WaitForExit();
            Console.WriteLine(cmd.StandardOutput.ReadToEnd());
        }

        public static void setPathToProcMon(string path)
        {
            procMonPath = path;
        }

        public static void setIsHasherDone(Boolean b)
        {
            isHasherDone = b;
        }
    }
}
```

E.3.4 Code for the reaction when the ransomware is detected

```
 using  System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotPOC
{
  class ActionTaker
  {

    static string pathToBackingFile = "";
    static int INDEXER = 0;
    static string BACKINGNAME = "";
    static HashSet<int> pID = new HashSet<int>();
    static string NAMEONTEST = "";
    static List<string> killedProcesses = new List<string>();
    private static Boolean killedFirstProcess = false;
    private static DateTime firstKilledProcessTime = new DateTime();

    //A change has been registered to a honeypot
    public static void honeypotChange(string path)
    {
      //Shut down procmon in order to get logfile
      ProcMon.procmonTerminator(pathToBackingFile, BACKINGNAME + INDEXER);

      INDEXER++;
      //Start up procmon with a new backingfile
      var cpmbf = new Thread(() => ProcMon.createProcmonBackingFile(pathToBackingFile, BACKINGNAME + INDEXER));
      cpmbf.Start();

      Thread.Sleep(3000);

      //Convert the PMLfile to CSV
      ProcMon.convertPMLfileToCSV(pathToBackingFile, BACKINGNAME + (INDEXER - 1) + ".PML", "convertedFile" + (INDEXER - 1) + ".CSV");

      bool hasCSVbeenWritten = false;
      Console.WriteLine("Path␣to␣CSV␣file:␣" + pathToBackingFile + "\\" + "convertedFile" + (INDEXER - 1) + ".CSV");

      //Wait for the conversion to be completed
      while (hasCSVbeenWritten == false)
```

```
    {
      try
      {
        using (Stream stream = new FileStream(pathToBackingFile + "\\" + "convertedFile" + (INDEXER - 1) + ".CSV", FileMode.Open))
        {
          hasCSVbeenWritten = true;
          stream.Dispose();
        }
      }
      catch (IOException)
      {

      }
      Thread.Sleep(50);
    }
    //Parse the CSVfile
    List<CSVfileHandler> parsedData = CSVfileHandler.CSVparser(pathToBackingFile + "\\" + "convertedFile" + (INDEXER - 1) + ".CSV");

    //Kill every process that has touched a honeypot
    foreach (var item in parsedData)
    {
      if (!item.processName.Equals("Explorer.EXE") || !item.processName.Equals("HoneyPotFilemon.exe"))
      {
        try
        {
          pID.Add(item.PID);
          killedProcesses.Add(Process.GetProcessById(item.PID).ProcessName);
          try
          {
            Console.WriteLine("Process: " + Process.GetProcessById(item.PID).ProcessName + " is killed due to suspicious behaviour");
            killProcess(item.PID);
          }
          catch (Exception)
          {
            //Save processname as a temp
            Console.WriteLine("Killing of the process failed");
          }
        }
        catch
        {

        }
      }
    }

    if (!killedFirstProcess)
    {
```

```
            firstKilledProcessTime = DateTime.Now;
            killedFirstProcess = true;
          }
        }

        private static void killProcess(int PID)
        {
          var process = Process.GetProcessById(PID);
          process.Kill();
          process.WaitForExit();
        }

        public static void setPathToBackingFile(string path)
        {
          pathToBackingFile = path;
        }

        public static void setBackingName(string name)
        {
          BACKINGNAME = name;
        }

        public static List<string> getKilledProcesses()
        {
          return killedProcesses;
        }

        public static DateTime getFirstKilledTime()
        {
          return firstKilledProcessTime;
        }

        public static void terminateProcmon()
        {
          ProcMon.procmonTerminator(pathToBackingFile, BACKINGNAME +
        INDEXER);
        }
      }
    }
```

E.3.5 Filemon for logging

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Threading;
using System.Collections;

namespace HoneyPotPOC.PocLogger

{
  class Filemon
  {

    static Dictionary<DateTime, string> fileMonChanges = new Dictionary<DateTime, string>();
    public static int i = 0;
    public static int temp = 0;
    public static Hashtable eventTimeLog = new Hashtable();
    private static Boolean stopAddingToLog = false;
    private static FileSystemWatcher watcher = new FileSystemWatcher();
    public static void CreateFileWatcher(string path)
    {
      //FileSystemWatcher can monitor changes in files

      //The given path dictates what directory the watcher will monitor
      watcher.Path = path;

      //The NotifyFilters determine what the monitors triggers upon.
      //It can also be a change in size.
      watcher.NotifyFilter = NotifyFilters.Size | NotifyFilters.LastWrite | NotifyFilters.FileName;

      //The filter gives the watcher a specific filename to look for
      // "*honeypot.*" monitors every file with honeypot in the ending, and every format.
      watcher.Filter = "*";

      //This tells the watcher when to react on different changes
      watcher.Created += new FileSystemEventHandler(OnChanged);
      watcher.Changed += new FileSystemEventHandler(OnChanged);
      watcher.Deleted += new FileSystemEventHandler(OnChanged);
      watcher.Renamed += new RenamedEventHandler(OnRenamed);

      watcher.EnableRaisingEvents = true;
      //IncludeSubdirectories does such that not only the directory given is monitored
      //but also every single subdirectory of the given directory
```

```
      watcher.IncludeSubdirectories = true;
    }


    //Event handeler if an object is changed
    private static void OnChanged(object source, FileSystemEventArgs e)
    {
      if (!stopAddingToLog)
      {
        if (!fileMonChanges.ContainsKey(DateTime.Now))
        {
          fileMonChanges.Add(DateTime.Now, e.FullPath);
        }
      }
    }


    //Event handeler if an object is renamed
    private static void OnRenamed(object source, RenamedEventArgs e)
    {
      if (!stopAddingToLog)
      {
        if (!fileMonChanges.ContainsKey(DateTime.Now))
        {
          fileMonChanges.Add(DateTime.Now, e.FullPath);
        }
      }
    }

    public static Dictionary<DateTime, string> getFilemonChanges()
    {
      return fileMonChanges;
    }

    //Stops adding to log such that an iteration won t trigger an error
    public static void setStopAddingToLog(Boolean b)
    {
      stopAddingToLog = b;
    }

    public static void setWatcherToStop()
    {
      watcher.EnableRaisingEvents = false;
    }
  }
}
```

E.3.6 Code for logging data

```
  using  HoneyPotFilemon.PocLogger;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace HoneyPotPOC.PocLogger
{
  class Logger
  {
    private static int INTERVALFORLOOP = 500;
    private static int MINUTESOFLOGGING = 25;
    private static string NAMEONTEST = "test";
    private static Boolean MONITORSTATUS = true;
    private static Boolean HASFETCHED = false;
    private static PerformanceCounter cpuUsageCounter;
    private static PerformanceCounter ramUsageCounter;
    private static PerformanceCounter harddiskUsageCounter;
    private static PerformanceCounter threadCounter;
    private static PerformanceCounter handleCounter;

    private static int amountOfLoops = 0;
    private static List<string> removeKeyList = new List<string>();
    private static List<string> changedKeyList = new List<string>();
    private static List<string> inStartDictionary = new List<string>();
    private static List<string> inEndDictionary = new List<string>();
    private static Dictionary<string, string>.KeyCollection hashedFilesAtStartKeys = null;
    private static Dictionary<string, string>.KeyCollection hashedFilesAtEndKeys = null;
    private static Dictionary<DateTime, string> fileMonChanges = Filemon.getFilemonChanges();
    private static List<float> cpuList = new List<float>();
    private static List<float> ramList = new List<float>();
    private static List<float> harddiskList = new List<float>();
    private static List<float> threadList = new List<float>();
    private static List<float> handleList = new List<float>();
    static string path1 = @"C:\Users\PoC\Desktop";
    static string path2 = @"C:\Users\PoC\Documents";
    static string path3 = @"C:\Users\PoC\Downloads";
    static string path4 = @"C:\Users\PoC\Videos";
    static string pathFileWatch = @"C:\Users\PoC";
```

```


    //Give the correct path for the hashed filesystem.
    //This includes giving the hasher the same path as the logger.
    static string hashedFilePath = @"C:\Software\";

    //static string path1 = @"C:\Users\viruseater1\Documents";
    //static string path2 = @"C:\Users\viruseater1\Desktop";
    //static string path3 = @"C:\Users\viruseater1\Downloads";
    //static string path4 = @"C:\Users\viruseater1\Videos";

    //Add the path to the ransomware downloader
    private static string ransomwareDownloaderPath = "";

    private static readonly HttpClient client = new HttpClient();

    public static Boolean LogWriter(string PATH)
    {

      cpuUsageCounter = new PerformanceCounter("Processor", "% Processor Time", "_Total");
      ramUsageCounter = new PerformanceCounter("Memory", "Available MBytes");
      harddiskUsageCounter = new PerformanceCounter("PhysicalDisk", "% Disk Time", "_Total");
      threadCounter = new PerformanceCounter("Process", "Thread Count", "_Total");
      handleCounter = new PerformanceCounter("Process", "Handle Count", "_Total");

      postPoCTaken();

      Dictionary<string, string> hashedFilesAtStart = new Dictionary<string, string>();

      //Get the hashed files from the txt file
      hashedFilesAtStart = testParseTXTfile(hashedFilePath);


      ProcMon.setIsHasherDone(true);
      amountOfLoops = 0;

      //After the hashed files has been read the ransomware is downloaded
      programExecuter.executeProgram(ransomwareDownloaderPath);


      var fw = new Thread(() => FileMon.createFileWatcher(pathFileWatch));
      fw.Start();

      var tmp = new Thread(() => Filemon.CreateFileWatcher(pathFileWatch));
      tmp.Start();
```

```
    //Find the start timestamp
    DateTime startTimeStamp = DateTime.Now;

    TimeSpan span = DateTime.Now.Subtract(startTimeStamp);

    //Loggs performance
    while (span.Minutes < MINUTESOFLOGGING)
    {
      amountOfLoops++;

      cpuList.Add(getCurrentCpuUsage());
      ramList.Add(getAvailableRAM());
      harddiskList.Add(getHarddiskUsage());
      threadList.Add(getThreadCount());
      handleList.Add(getHandleCount());

      Thread.Sleep(INTERVALFORLOOP);


      span = DateTime.Now.Subtract(startTimeStamp);
    }

    Filemon.setStopAddingToLog(true);
    fileMonChanges = Filemon.getFilemonChanges();

    Filemon.setWatcherToStop();
    FileMon.setWatcherToStop();
    ActionTaker.terminateProcmon();

    //Combines the hashed files from the four directories into one
    Dictionary<string, string> hashedFilesAtEnd = new Dictionary<string, string>();
    Dictionary<string, string> hashedFilesAtEndtemp1 = new Dictionary<string, string>();
    Dictionary<string, string> hashedFilesAtEndtemp2 = new Dictionary<string, string>();
    Dictionary<string, string> hashedFilesAtEndtemp3 = new Dictionary<string, string>();
    Dictionary<string, string> hashedFilesAtEndtemp4 = new Dictionary<string, string>();
    Hasher tempEndHasher1 = new Hasher();
    hashedFilesAtEndtemp1 = tempEndHasher1.fileHasher(path1);

    Hasher tempEndHasher2 = new Hasher();
    hashedFilesAtEndtemp2 = tempEndHasher2.fileHasher(path2);

    Hasher tempEndHasher3 = new Hasher();
    hashedFilesAtEndtemp3 = tempEndHasher3.fileHasher(path3);

    Hasher tempEndHasher4 = new Hasher();
    hashedFilesAtEndtemp4 = tempEndHasher4.fileHasher(path4);
```

```
hashedFilesAtEndtemp1.ToList().ForEach(x => hashedFilesAtEnd.Add(x.Key, x.Value));
hashedFilesAtEndtemp2.ToList().ForEach(x => hashedFilesAtEnd.Add(x.Key, x.Value));
hashedFilesAtEndtemp3.ToList().ForEach(x => hashedFilesAtEnd.Add(x.Key, x.Value));
hashedFilesAtEndtemp4.ToList().ForEach(x => hashedFilesAtEnd.Add(x.Key, x.Value));


//Find the end timestamp
DateTime endTimeStamp = DateTime.Now;

//Figure out what has changed.
removeKeyList = new List<string>();
changedKeyList = new List<string>();
inStartDictionary = new List<string>();
inEndDictionary = new List<string>();
foreach (var item in hashedFilesAtStart)
{
  if (hashedFilesAtEnd.ContainsKey(item.Key))
  {
    if (hashedFilesAtStart[item.Key].Equals(hashedFilesAtEnd[item.Key]))
    {
      removeKeyList.Add(item.Key);
    }
    else
    {
      changedKeyList.Add(item.Key);
    }
  }
  else
  {
    inStartDictionary.Add(item.Key);
  }
}
//Removing non changed duplicates
for (int i = 0; i < removeKeyList.Count; i++)
{
  hashedFilesAtStart.Remove(removeKeyList[i]);
  hashedFilesAtEnd.Remove(removeKeyList[i]);
}
for (int i = 0; i < changedKeyList.Count; i++)
{
  hashedFilesAtStart.Remove(changedKeyList[i]);
  hashedFilesAtEnd.Remove(changedKeyList[i]);
}
//Finding files that has been created since start
foreach (var item in hashedFilesAtEnd)
{
  if (!hashedFilesAtStart.ContainsKey(item.Key))
  {
    inEndDictionary.Add(item.Key);
  }
```

```
      }
      hashedFilesAtStartKeys = hashedFilesAtStart.Keys;
      hashedFilesAtEndKeys = hashedFilesAtEnd.Keys;
      return true;
    }

    private static float getCurrentCpuUsage()
    {
      return cpuUsageCounter.NextValue();
    }

    private static float getAvailableRAM()
    {
      return ramUsageCounter.NextValue();
    }

    private static float getHarddiskUsage()
    {
      return harddiskUsageCounter.NextValue();
    }

    private static float getThreadCount()
    {
      return threadCounter.NextValue();
    }

    private static float getHandleCount()
    {
      return handleCounter.NextValue();
    }

    public static async void postPoCPosted()
    {
      string cpuReturn = returnMonitorListAsString(cpuList);
      string ramReturn = returnMonitorListAsString(ramList);
      string harddiskReturn = returnMonitorListAsString(harddiskList);
      string threadReturn = returnMonitorListAsString(threadList);
      string handleReturn = returnMonitorListAsString(handleList);
      List<string> killedProcesses = ActionTaker.getKilledProcesses();

      string changedFilesReturn = "";
      string deletedFilesReturn = "";
      string newFilesReturn = "";
      string filemonChangesReturn = "";
      string killedProcessesReturn = "";


      for (int i = 0; i < changedKeyList.Count - 1; i++)
      {
        changedFilesReturn += changedKeyList[i];
        changedFilesReturn += "?";
```

```
    }
    foreach (string s in hashedFilesAtStartKeys)
    {
      deletedFilesReturn += s;
      deletedFilesReturn += "?";
    }
    foreach (string s in hashedFilesAtEndKeys)
    {
      newFilesReturn += s;
      newFilesReturn += "?";
    }
    foreach (var item in fileMonChanges)
    {
      filemonChangesReturn += item.Value + ":" + item.Key.ToString("dd/MM/yyyy HH:mm:ss.fff");
      filemonChangesReturn += "?";
    }
    foreach (string s in killedProcesses)
    {
      killedProcessesReturn += s;
      killedProcessesReturn += "?";
    }

    var options = new
    {
      RansomwareName = NAMEONTEST,
      MonitorStatus = "1",
      MonitorCount = amountOfLoops.ToString(),
      CountChangedFiles = changedKeyList.Count().ToString(),
      CountDeletedFiles = hashedFilesAtStartKeys.Count().ToString(),
      CountNewFiles = hashedFilesAtEndKeys.Count().ToString(),
      CountFilemonObservations = fileMonChanges.Count().ToString(),
      CPU = cpuReturn,
      RAM = ramReturn,
      HDD = harddiskReturn,
      ThreadCount = threadReturn,
      HandleCount = handleReturn,
      ListChangedFiles = changedFilesReturn,
      ListDeletedFiles = deletedFilesReturn,
      ListNewFiles = newFilesReturn,
      ListFilemonObservations = filemonChangesReturn,
      NameOfShutdownRansomware = killedProcessesReturn,
      Detected = FileMon.getFirstDetected().ToString("dd/MM/yyyy HH:mm:ss.fff"),
      Shutdown = ActionTaker.getFirstKilledTime().ToString("dd/MM/yyyy HH:mm:ss.fff")
    };


    var stringPayload = JsonConvert.SerializeObject(options);
    var content = new StringContent(stringPayload, Encoding.UTF8, "application/json");

```

```
    var response = client.PostAsync("http://192.168.8.102/v1/index.php/posthp1posted", content).Result;
    var result = await response.Content.ReadAsByteArrayAsync();
  }

  public static void getPoCRansomware()
  {
    var responseString = client.GetStringAsync("http://192.168.8.102/v1/index.php/gethp1ransomware").Result;

    NAMEONTEST = findNAMEONTEST(responseString);
    Console.WriteLine(NAMEONTEST);
  }

  public static async void postPoCFetched()
  {
    var values = new Dictionary<string, string>
    {
      {"RansomwareName", NAMEONTEST}
    };

    var content = new FormUrlEncodedContent(values);

    var response = client.PostAsync("http://192.168.8.102/v1/index.php/posthp1fetched", content).Result;

    HASFETCHED = true;

    var responseString = await response.Content.ReadAsByteArrayAsync();
  }


  private static string findNAMEONTEST(string responsestring)
  {
    int i = 0;
    int j = 0;
    foreach (char c in responsestring)
    {
      if (i == 5)
      {
        return responsestring.Substring(j, responsestring.Length - j - 4);
      }
      if (c.Equals ( " ) )
      {
        i++;
      }
      j++;
    }

    return "Could Not Find";
  }

```

```
    private␣static␣string␣returnMonitorListAsString(List<float>␣convertedList)
    {
      string␣temp␣=␣"";
      for␣(int␣i␣=␣0;␣i␣<␣amountOfLoops;␣i++)
      {
        temp␣+=␣convertedList[i].ToString();
        temp␣+=␣"?";
      }
      return␣temp;
    }

    public␣static␣async␣void␣postPoCTested()
    {
      var␣values␣=␣new␣Dictionary<string,␣string>
      {
        {"RansomwareName", NAMEONTEST}
      };

      var␣content␣=␣new␣FormUrlEncodedContent(values);

      var␣response␣=␣client.PostAsync("http://192.168.8.102/v1/index.php/posthp1tested", content).Result;

      var responseString = await response.Content.ReadAsByteArrayAsync();
    }

    public static async void postPoCTaken()
    {
      var values = new Dictionary<string, string>
      {
        {"RansomwareName", NAMEONTEST}
      };

      var content = new FormUrlEncodedContent(values);

      var response = client.PostAsync("http://192.168.8.102/v1/index.php/posthp1taken", content).Result;

      var responseString = await response.Content.ReadAsByteArrayAsync();
    }


    public static string getNAMEONTEST()
    {
      return NAMEONTEST;
    }

    public static void setRansomwareDownloaderPath(string s)
    {
      ransomwareDownloaderPath = s;
    }

```

```
    public static Boolean getHasFetched()
    {
      return HASFETCHED;
    }

    public static Dictionary<string, string> testParseTXTfile(
    string hashedFilePath)
    {
      string line;
      string[] pairs = new string[2];
      Dictionary<string, string> hashedFilesReturn = new
    Dictionary<string, string>();
      System.IO.StreamReader file =
        new System.IO.StreamReader(hashedFilePath + "\\
    HashedFilesLog.txt");
      while ((line = file.ReadLine()) != null)
      {
        pairs = line.Split( ? );
        hashedFilesReturn.Add(pairs[0], pairs[1]);
      }
      file.Close();

      return hashedFilesReturn;
    }
  }
}
```

E.4 Shannon Entropy Prove of Concept

This appendix shows the shannon entropy detection method, most of the code is similar to the one from Honeypot, therefore only the code that differs is added.

E.4.1 Main control unit

```
 using  ShannonPOC.ShannonLogger;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ShannonPOC
{
  class Program
  {
    //Set path for folders
    private static string path1 = @"C:\Users\Baseline\Desktop";
    private static string path2 = @"C:\Users\Baseline\Documents";
    private static string path3 = @"C:\Users\Baseline\Downloads";
    private static string path4 = @"C:\Users\Baseline\Videos";

    //Set path for procmon
    static string PATH = @"C:\Users\Baseline";
    static string BACKINGNAME = "backingFromProcMon";
    static string pathToBackingFile = @"C:\procmon\
    backingFileTest";
    static string ProcMonPath = @"C:\procmon\Procmon.exe";

    static string RANSOMWAREDOWNLOADERPATH = @"C:\Software\
    ShannonRansomwareDownloader\bin\Release\
    ShannonRansomwareDownloader.exe";

    //Set threshold and duration
    static double entropyThreshold = 0.9;
    static int thresholdToReaction = 2;
    static int secondsInThreshold = 60;

    static void Main(string[] args)
    {
      //Wait for program to start
      Thread.Sleep(30000);

      shannonEntropyFileMonDetection();

    }

```

```
    public static void shannonEntropyFileMonDetection()
    {
      //Get name of ransomware
      Logger.getPoCRansomware();

      Thread.Sleep(1000);

      //Post name to server that the ransomware has been fetched
      Logger.postPoCFetched();

      //Wait for the server to respond
      while (!Logger.getHasFetched())
      {
        Thread.Sleep(500);
      }

      //Initialize variables
      Logger.setRansomwareDownloaderPath(RANSOMWAREDOWNLOADERPATH
    );

      ActionTaker.setBackingName(BACKINGNAME);
      ActionTaker.setPathToBackingFile(pathToBackingFile);

      ProcMon.setPathToProcMon(ProcMonPath);

      FilemonEventHandler.setEntropyThreshold(entropyThreshold);
      FilemonEventHandler.setThresholdToReaction(
    thresholdToReaction);
      FilemonEventHandler.setSecondsInThreshold(
    secondsInThreshold);

      Logger.setPath1(path1);
      Logger.setPath2(path2);
      Logger.setPath3(path3);
      Logger.setPath4(path4);
      Logger.setPathFileWatch(PATH);

      //Find entropy of all files
      ShannonEntropy temp1 = new ShannonEntropy();
      temp1.getEntropyOfAllFilesInPath(path1);

      ShannonEntropy temp2 = new ShannonEntropy();
      temp2.getEntropyOfAllFilesInPath(path2);

      ShannonEntropy temp3 = new ShannonEntropy();
      temp3.getEntropyOfAllFilesInPath(path3);

      ShannonEntropy temp4 = new ShannonEntropy();
      temp4.getEntropyOfAllFilesInPath(path4);

      //Print the entropies
      Dictionary<string, double> test = ShannonEntropy.
    getSavedEntropies();
      foreach (var item in test)
      {
```

```
        Console.WriteLine(item.Key + " - " + item.Value);
      }


      //Start procmon
      BACKINGNAME = BACKINGNAME + 0;
      var t = new Thread(() => ProcMon.createProcmonBackingFile(pathToBackingFile, BACKINGNAME));
      t.Start();

      //Start filemon
      //When filemon sees a reaction it posts to filemoneventhandler
      //Filemoneventhandler then deems if it is nessesary to take action, using actiontaker
      Console.WriteLine(Logger.getNAMEONTEST());

      //Start logger
      Logger.LogWriter(PATH);

      //Post to server that it has been tested
      Logger.postPoCTested();

      //Post to server the results
      Logger.postPoCPosted();

      Thread.Sleep(30000);

    }
  }
}
```

E.4.2 Filemon for shannon entropy

```
 using  System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Threading;
using System.Collections;

namespace ShannonPOC
{
  class FileMon
  {
    private static FileSystemWatcher watcher = new FileSystemWatcher();

    public static void CreateFileWatcher(string path)
    {
      //FileSystemWatcher can monitor changes in files

      //The given path dictates what directory the watcher will monitor
      watcher.Path = path;

      //The NotifyFilters determine what the monitors triggers upon.
      //It can also be a change in size.
      watcher.NotifyFilter = NotifyFilters.Size | NotifyFilters.LastWrite | NotifyFilters.FileName;

      //The filter gives the watcher a specific filename to look for
      // "*honeypot.*" monitors every file with honeypot in the ending, and every format.
      watcher.Filter = "*";

      //This tells the watcher when to react on different changes
      watcher.Created += new FileSystemEventHandler(OnChanged);
      watcher.Changed += new FileSystemEventHandler(OnChanged);
      watcher.Deleted += new FileSystemEventHandler(OnChanged);
      watcher.Renamed += new RenamedEventHandler(OnRenamed);

      watcher.EnableRaisingEvents = true;
      //IncludeSubdirectories does such that not only the directory given is monitored
      //but also every single subdirectory of the given directory
      watcher.IncludeSubdirectories = true;
    }


    //Event handeler if an object is changed
    private static void OnChanged(object source, FileSystemEventArgs e)
```

```
    {
      //Cancel out appdata
      Console.WriteLine(e.FullPath + " is " + e.ChangeType);

      if (e.FullPath.Contains(@"C:\Users\Baseline\Desktop")
        || e.FullPath.Contains(@"C:\Users\Baseline\Documents")
        || e.FullPath.Contains(@"C:\Users\Baseline\Downloads")
        || e.FullPath.Contains(@"C:\Users\Baseline\Videos"))
      {
        if (e.FullPath.Contains("."))
        {
          if (e.ChangeType.ToString().Equals("Changed"))
          {
            FilemonEventHandler.changeOccured(e);
          }
          else if (e.ChangeType.ToString().Equals("Created"))
          {
            FilemonEventHandler.creationOccured(e);
          }
          else if (e.ChangeType.ToString().Equals("Deleted"))
          {
            FilemonEventHandler.deletionOccured(e);
          }
        }
      }
    }


    //Event handler if an object is renamed
    private static void OnRenamed(object source, RenamedEventArgs e)
    {
      Console.WriteLine(e.OldFullPath + " is renamed to " + e.FullPath);
      if (e.OldFullPath.Contains(@"C:\Users\Baseline\Desktop")
        || e.OldFullPath.Contains(@"C:\Users\Baseline\Documents")
        || e.OldFullPath.Contains(@"C:\Users\Baseline\Downloads")
        || e.OldFullPath.Contains(@"C:\Users\Baseline\Videos"))
      {
        if (ShannonEntropy.getSavedEntropies().ContainsKey(e.OldFullPath))
        {
          Double tempEntropy = ShannonEntropy.getSavedEntropies()[e.OldFullPath];
          ShannonEntropy.removeKeyFromSavedEntropies(e.OldFullPath);
          ShannonEntropy.addKeyAndDoubleToSavedEntropies(e.FullPath, tempEntropy);
        }
      }
    }


    public static void setWatcherToStop()
    {
```

```
            Console.WriteLine("Filemon not logging ransomwares anymore"
        );
            watcher.EnableRaisingEvents = false;
        }
    }
}
```

E.4.3 Event handler for filemon events

```
 using  System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ShannonPOC
{
  class FilemonEventHandler
  {

    private static DateTime firstDetected;
    static Boolean hasMadeFirstDetection = false;
    private static double entropyThreshold = 0.0;
    private static int thresholdToReaction = 0;
    private static List<DateTime> threshold = new List<DateTime>();
    private static int secondsInThreshold = 0;

    internal static void changeOccured(FileSystemEventArgs e)
    {
      //Kig p  entropien f r og efter
      Dictionary<string, double> savedEntropies = ShannonEntropy.getSavedEntropies();
      FileInfo changedFile = new FileInfo(e.FullPath);
      ShannonEntropy entropyCalculator = new ShannonEntropy();
      Double changedFileEntropy = entropyCalculator.CalculateEntropy(changedFile);
      Double originalFileEntropy = 0.0;

      Console.WriteLine("File " + e.FullPath + " has been changed to and has now an entropy of " + changedFileEntropy);
      if (changedFileEntropy == -1)
      {
        return;
      }

      try
      {
        originalFileEntropy = savedEntropies[e.FullPath];
      }
      catch (Exception)
      {

      }

      entropyHandler(e, originalFileEntropy, changedFileEntropy);
    }

    internal static void creationOccured(FileSystemEventArgs e)
    {
```

```
    //Er der en fil i directoriet der har samme entropi som denne er den blot rykket
    //L b listen af keys igennem, se value, nogen ens? Godt
    //add til databasen den nye fil, slet den gamle

    Dictionary<string, double> savedEntropies = new Dictionary<string, double>();

    savedEntropies = ShannonEntropy.getSavedEntropies();

    FileInfo createdFileInfo = new FileInfo(e.FullPath);

    ShannonEntropy entropyCreator = new ShannonEntropy();
    double createdFileEntropy = entropyCreator.CalculateEntropy(createdFileInfo);


    Console.WriteLine("File " + e.FullPath + " has been created and entropy is now " + createdFileEntropy);
    if (createdFileEntropy == -1)
    {
      return;
    }

    Boolean fileHasBeenMoved = false;
    string oldFilePath = "";

    foreach (var item in savedEntropies)
    {
      if(item.Value == createdFileEntropy)
      {
        //File has been moved
        fileHasBeenMoved = true;
        oldFilePath = item.Key;
      }
    }

    if (fileHasBeenMoved)
    {
      ShannonEntropy.removeKeyFromSavedEntropies(oldFilePath);
      ShannonEntropy.addKeyAndDoubleToSavedEntropies(e.FullPath, createdFileEntropy);
    }
    else
    {
      //TODO find threshold p   nye filer og om entropien er for h j
      ShannonEntropy.removeKeyFromSavedEntropies(oldFilePath);
      ShannonEntropy.addKeyAndDoubleToSavedEntropies(e.FullPath, createdFileEntropy);
      if(createdFileEntropy > entropyThreshold)
      {
        react(e);
      }
    }
```

```
    }

    internal static void deletionOccured(FileSystemEventArgs e)
    {
      string[] filesInDirectory = null;

      filesInDirectory = Directory.GetFiles(returnFilePath(e.FullPath));

      Boolean newSimilarFileIsCreated = false;

      ShannonEntropy entropyCreator = new ShannonEntropy();

      string fileName = returnFileName(e.FullPath);

      double oldEntropy = ShannonEntropy.getSavedEntropies()[e.FullPath];
      foreach (string s in filesInDirectory)
      {
        if (s.Contains(fileName))
        {
          newSimilarFileIsCreated = true;
          FileInfo newFileInfo = new FileInfo(s);
          double newEntropy = entropyCreator.CalculateEntropy(newFileInfo);

          //TODO react if needed
          entropyHandler(e, oldEntropy, newEntropy);
        }
      }

      ShannonEntropy.removeKeyFromSavedEntropies(e.FullPath);
    }

    private static void react(FileSystemEventArgs e)
    {
      threshold.Add(DateTime.Now);
      List<DateTime> temp = new List<DateTime>();
      DateTime now = DateTime.Now;

      foreach (DateTime t in threshold)
      {
        if (secondsInThreshold < (now.Subtract(t).Seconds))
        {
          temp.Add(t);
        }
      }

      foreach (DateTime t in temp)
      {
        threshold.Remove(t);
      }

      if (threshold.Count > thresholdToReaction)
      {
```

```
        if (!hasMadeFirstDetection)
        {
          hasMadeFirstDetection = true;
          firstDetected = DateTime.Now;
        }
        Console.WriteLine("File: " + e.FullPath + " has been " + e.ChangeType + " and the responsible process will now pay the ultimate price!");

        ActionTaker.shannonReaction(e.FullPath);
      }
    }

    private static void entropyHandler(FileSystemEventArgs e, double originalFileEntropy, double newFileEntropy)
    {
      if(originalFileEntropy < 0.1)
      {
        if(newFileEntropy > 0.6)
        {
          react(e);
        }
      }
      else if (originalFileEntropy < 0.2)
      {
        if(newFileEntropy > 0.65)
        {
          react(e);
        }
      }
      else if (originalFileEntropy < 0.3)
      {
        if (newFileEntropy > 0.65)
        {
          react(e);
        }
      }
      else if (originalFileEntropy < 0.4)
      {
        if (newFileEntropy > 0.7)
        {
          react(e);
        }
      }
      else if (originalFileEntropy < 0.5)
      {
        if (newFileEntropy > 0.7)
        {
          react(e);
        }
      }
      else if (originalFileEntropy < 0.6)
      {
        if (newFileEntropy > 0.8)
        {
```

```
            react(e);
          }
        }
        else if (originalFileEntropy < 0.7)
        {
          if (newFileEntropy > 0.8)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.8)
        {
          if (newFileEntropy > 0.85)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.9)
        {
          if (newFileEntropy > 0.95)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.91)
        {
          if (newFileEntropy > 0.97)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.92)
        {
          if (newFileEntropy > 0.97)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.93)
        {
          if (newFileEntropy > 0.975)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.94)
        {
          if (newFileEntropy > 0.98)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.95)
        {
          if (newFileEntropy > 0.98)
```

```
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.96)
        {
          if (newFileEntropy > 0.985)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.97)
        {
          if (newFileEntropy > 0.99)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.98)
        {
          if (newFileEntropy > 0.99)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.99)
        {
          if (newFileEntropy > 0.995)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.999)
        {
          if (newFileEntropy > 0.9992)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 0.9999)
        {
          if (newFileEntropy > 0.9999)
          {
            react(e);
          }
        }
        else if (originalFileEntropy < 1)
        {
          if (newFileEntropy > 0.99995)
          {
            react(e);
          }
        }
      }
```

```

    public static string returnFileName(string fullPath)
    {

      int lastSlash = 0;
      int lastDot = 0;
      string fileName = "";

      for (int i = 0; i < fullPath.Length - 1; i++)
      {
        if (fullPath.Substring(i, 1).Equals(@"\"))
        {
          lastSlash␣=␣i;
        }
        if␣(fullPath.Substring(i,␣1).Equals("."))
        {
          lastDot␣=␣i;
        }
      }
      fileName␣=␣fullPath.Substring(lastSlash␣+␣1,␣lastDot␣-␣
    lastSlash␣-␣1);

      return␣fileName;
    }

    public␣static␣string␣returnFilePath(string␣fullPath)
    {

      int␣lastSlash␣=␣0;
      int␣lastDot␣=␣0;
      string␣fileName␣=␣"";

      for␣(int␣i␣=␣0;␣i␣<␣fullPath.Length␣-␣1;␣i++)
      {
        if␣(fullPath.Substring(i,␣1).Equals(@"\"))
        {
          lastSlash␣=␣i;
        }
      }
      fileName␣=␣fullPath.Substring(0,lastSlash␣+␣1);

      return␣fileName;
    }

    internal␣static␣DateTime␣getFirstDetected()
    {
      return␣firstDetected;
    }
    public␣static␣void␣setFirstDetected()
    {
      firstDetected␣=␣DateTime.Now;
    }

    public␣static␣void␣setEntropyThreshold(double␣d)
    {
```

```
      entropyThreshold = d;
    }

    public static void setThresholdToReaction(int i)
    {
      thresholdToReaction = i;
    }

    public static void setSecondsInThreshold(int i)
    {
      secondsInThreshold = i;
    }
  }
}
```

E.4.4 Shannon entropy calculator

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ShannonPOC
{
  class ShannonEntropy
  {
    private static Dictionary<string, double> savedEntropies = new Dictionary<string, double>();

    public Dictionary<string, double> getEntropyOfAllFilesInPath(string path)
    {
      string[] filesInDirectory = null;
      Console.WriteLine(path);

      //Check if it is possible to get the files in path, if not return findings
      try
      {
        filesInDirectory = Directory.GetFiles(path);
      }
      catch (Exception)
      {
        return savedEntropies;
      }

      //Takes the entropy of each file in directory
      FileInfo tempFil;
      foreach (string file in filesInDirectory)
      {
        tempFil = new FileInfo(file);
        savedEntropies.Add(file, CalculateEntropy(tempFil));
      }

      //Get every subdirectory in the given path
      var subDirectories = Directory.GetDirectories(path);

      //Iterates though the subdirectories
      foreach (var directory in subDirectories)
      {
        //Creates a string with the name of the subdirectory only
        string dirName = new DirectoryInfo(directory).Name;

        //Calls the function itself for every subdirectory
        getEntropyOfAllFilesInPath(path + "\\" + dirName);
      }

      return savedEntropies;
```

```
    }

    public double CalculateEntropy(FileInfo file)
    {
      //Set the range to 256
      int range = byte.MaxValue + 1;

      //Read the bytes of the file into a byte array
      //If the path is not a file but a directory it returns 0
      byte[] values;
      try
      {
        values = File.ReadAllBytes(file.FullName);
      }
      catch (Exception)
      {
        return -1;
      }

      //Make a long array the size of the range we are interested in
      long[] counts = new long[range];
      foreach (byte value in values)
      {
        //Count how many occurences there are of each byte
        counts[value] = counts[value] + 1;
      }

      double entropy = 0;

      //Adds the entropy of every single number in the values array together
      foreach (long count in counts)
      {
        if(count != 0)
        {
          double probability = (double)count / values.LongLength;
          entropy -= probability * Math.Log(probability, range);
        }
      }
      return entropy;
    }

    public static Dictionary<string, double> getSavedEntropies()
    {
      return savedEntropies;
    }

    public static void removeKeyFromSavedEntropies(string key)
    {
      if (savedEntropies.ContainsKey(key))
      {
        savedEntropies.Remove(key);
      }
      else
```

```
            {
                Console.WriteLine("Could not remove key "+ key +" since it does not exist in the list");
            }
        }

        public static void addKeyAndDoubleToSavedEntropies(string key, double value)
        {
            if (!savedEntropies.ContainsKey(key))
            {
                savedEntropies.Add(key, value);
            }
            else
            {
                Console.WriteLine("Could not add key " + key + " to the list since it is already there");
            }
        }
    }
}
```

E.5 Practical tools for extracting data

The following is code made for collecting every test for a single detection method and storing it into seperate text documents, one for each ransomware tested upon.

E.5.1 Main control unit

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseCollector
{
  class Program
  {
    //Set variables for collecting data
    static string databaseinputbase = "http://192.168.8.102/v1/index.php/getdata";
    static string databaseTester = "hp1";
    static string middlepart = "?RansomwareName=";
    static string ransomwareName = "Vipsana2";

    //Give path to files and folders
    static string fileToVirusNames = @"RansomwareList.txt";
    static string pathToFolders = @"C:\Speciale\Relevant Data";

    static void Main(string[] args)
    {

      //Read txt file with all virus name

      List<string> listOfRansomwareNames = VirusFileParser.parseTxtToList(fileToVirusNames);
      string ransomwareOutput = "";
      foreach (var item in listOfRansomwareNames)
      {
        Console.WriteLine(item.Substring(1, item.Length - 2));
        ransomwareName = item.Substring(1,item.Length-2);
        //Get ransomware data from server
        ransomwareOutput = ServerCommunicator.returnDatabaseOutputForRansomware(databaseinputbase + databaseTester + middlepart + ransomwareName);

        //Create a file for the given ransomware
```

```
                ServerOutputHandler.CreateReadableFileForRansomware(
        databaseTester,ransomwareName,ransomwareOutput,pathToFolders)
        ;
            }

            Console.ReadLine();
        }
    }
}
```

E.5.2 Handling of the output from server

```
 using  System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseCollector
{
  class ServerOutputHandler
  {
    internal static void CreateReadableFileForRansomware(string
    databaseTester, string ransomwareName, string
    ransomwareOutput, string path)
    {
      string firstColon = "";
      string secondColon = "";
      int firstColonPos = 0;
      int secondColonPos = 0;
      string data = "";

      List<string> serverOutput = new List<string>();

      for (int i = 0; i < ransomwareOutput.Length -1; i++)
      {
        if(firstColonPos == 0)
        {
          firstColon = ransomwareOutput.Substring(i, 1);
        }
        else if(secondColonPos == 0)
        {
          secondColon = ransomwareOutput.Substring(i, 1);
        }
        else
        {

        }
        if(firstColon == "\"")
        {
          firstColonPos = i;
          firstColon = "";
        }
        if (secondColon == "\"")
        {
          secondColonPos = i;
          secondColon = "";
        }

        if(firstColonPos != 0 && secondColonPos != 0)
        {

          if (data.Equals("listFilemonObservations"))
          {
```

```
            data = ransomwareOutput.Substring(firstColonPos + 1, secondColonPos - firstColonPos - 1);
            data = fixFileMonObservations(data);
            serverOutput.Add(data);
          }
          else
          {
            data = ransomwareOutput.Substring(firstColonPos+1, secondColonPos - firstColonPos -1);
            serverOutput.Add(data);
          }

          firstColonPos = 0;
          secondColonPos = 0;
        }

      }


      string filePath = path + @"\"␣+␣databaseTester␣+␣@"\"␣+␣ransomwareName␣+␣".txt";
      Console.WriteLine(filePath);
      Console.WriteLine(path);
      if␣(!File.Exists(filePath))
      {
        //␣Create␣a␣file␣to␣write␣to.
        using␣(StreamWriter␣sw␣=␣File.CreateText(filePath))
        {
          foreach␣(var␣item␣in␣serverOutput)
          {
            sw.WriteLine(item);
          }
        }
      }
    }

    private␣static␣string␣fixFileMonObservations(string␣data)
    {
      int␣dataLength␣=␣data.Length;
      for␣(int␣i␣=␣0;␣i␣<␣dataLength-5;␣i++)
      {
        if␣(data.Substring(i,␣5).Equals("-2017"))
        {
          data␣=␣data.Substring(0,␣i␣-␣6)␣+␣"*"␣+␣data.Substring(i␣-␣5);
        }

      }
      return␣data;
    }
  }
}
```

Bibliography

[16] "An ISTR Special Report: Ransomware and Buisnesses 2016". In: *Ransomware and Businesses* (2016). URL: http://www.symantec.com/content/en/us/enterprise/media/security%7B%5C_%7Dresponse/whitepapers/ISTR2016%7B%5C_%7DRansomware%7B%5C_%7Dand%7B%5C_%7DBusinesses.pdf.

[AGM15] Magnus Almgren, Vincenzo Gulisano, and Federico Maggi. "Detection of intrusions and malware, and vulnerability assessment: 12th International conference, DIMVA 2015 Milan, Italy, july 9???10, 2015 proceedings". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9148 (2015), pp. 3–24. ISSN: 16113349. DOI: 10.1007/978-3-319-20550-2.

[AS16] M M Ahmadian and H R Shahriari. "2entFOX: A framework for high survivable ransomwares detection". In: *2016 13th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)* (2016), pp. 79–84. DOI: 10.1109/ISCISC.2016.7736455.

[Bou] Boutell.com. *WWW FAQs: What is the maximum length of a URL?* URL: https://boutell.com/newfaq/misc/urllength.html.

[Bow+] Brian M Bowen et al. "Baiting Inside Attackers Using Decoy Documents". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EEB4EFDA3C6E77A9BE7751E6BAF1973D?doi=10.1.1.150.1361%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[Bra] Russell Brandom. *UK hospitals hit with massive ransomware attack - The Verge*. URL: https://www.theverge.com/2017/5/12/15630354/nhs-hospitals-ransomware-hack-wannacry-bitcoin (visited on 06/23/2017).

[Che] Checkpoint. *Ransomware: A new approach to identifying, blocking and real-time remediation*. URL: https://www.checkpoint.com/downloads/product-related/whitepapers/wp-ransomware-identify-block-and-remedy.pdf.

[Cim] Catalin Cimpanu. *Xorist Ransomware Family Is Now Decryptable for Free.* URL: http://news.softpedia.com/news/xorist-ransomware-family-is-now-decryptable-for-free-502036.shtml (visited on 06/21/2017).

[Cle] CleanSofts. *Top 50 Most Popular Software.* URL: http://cleansofts.org/software/popular.html.

[Cuc] Cuckoo. *Cuckoo Sandbox.* URL: https://cuckoosandbox.org.

[Dat16] Datto. "Datto's State of the Channel Ransomware Report 2016". In: (2016). URL: http://pages.datto.com/rs/572-ZRG-001/images/DattoStateOfTheChannelRansomwareReport2016_RH.pdf.

[Edi] Sophos News Editor. *The current state of ransomware: CryptoWall – Sophos News.* URL: https://news.sophos.com/en-us/2015/12/17/the-current-state-of-ransomware-cryptowall/ (visited on 05/14/2017).

[End] Endgame. *WCry/WanaCry Ransomware Technical Analysis.* URL: https://www.endgame.com/blog/technical-blog/wcrywanacry-ransomware-technical-analysis.

[Fin] Jim Finkle. *Ransomware: Extortionist hackers borrow customer-service tactics | Reuters.* URL: http://www.reuters.com/article/us-usa-cyber-ransomware-idUSKCN0X917X (visited on 04/15/2017).

[Gam] Brianna Gammons. *The Psychology of Ransomware: 5 Mind Games Criminals Play.* URL: https://blog.barkly.com/ransomware-scare-tactics-mind-games.

[Hay] Kaoru Hayashi. *Trojan.Archiveus Technical Details | Symantec.* URL: https://www.symantec.com/security_response/writeup.jsp?docid=2006-050601-0940-99&tabid=2 (visited on 06/21/2017).

[Hig97] Harold Joseph Highland. "A History Of Computer Viruses -The Famous Trio". In: *Computers and Security, 16.* Ed. by Eugene H. Spafford. Vol. 16. Lecture Notes in Computer Science. 1997, pp. 416–429. ISBN: 0167-4048.

[Hos15] Diane Duros Hosfelt. "Automated detection and classification of cryptographic algorithms in binary programs through machine learning". In: (Mar. 2015). arXiv: 1503.01186. URL: http://arxiv.org/abs/1503.01186.

[Hyp11] Mikko Hypponen. *The History and the Evolution of Computer Viruses.* 2011. URL: https://archive.org/details/DEFCON_19_The_History_and_the_Evolution_of_Computer_Viruses.

[IBM] IBM. *Ransomware: How consumers and businesses value their data.* URL: https://www-03.ibm.com/press/us/en/pressrelease/51230.wss (visited on 06/21/2017).

[Int] Intel. *Intel Core i5-660 vs i7-3770S*. URL: http://www.cpu-world.com/Compare/465/Intel%7B%5C_%7DCore%7B%5C_%7Di5%7B%5C_%7Di5-660%7B%5C_%7Dvs%7B%5C_%7DIntel%7B%5C_%7DCore%7B%5C_%7Di7%7B%5C_%7Di7-3770S.html (visited on 06/23/2017).

[Kas] Michael Kassner. *Ransomware: Extortion via the Internet - TechRepublic*. URL: http://www.techrepublic.com/blog/it-security/ransomware-extortion-via-the-internet/ (visited on 05/14/2017).

[Kor] Jesse Kornblum. "Identifying almost identical files using context triggered piecewise hashing". In: (). DOI: 10.1016/j.diin.2006.06.015. URL: http://production.datastore.cvt.dk/filestore?oid=539cd4e560ad71dd2500f98e%7B%5C&%7Dtargetid=539cd4e560ad71dd2500f990.

[Lab] Kaspersky Lab. *WannaCry mistakes that can help you restore files after infection*. URL: https://thehackernews.com/2017/06/wannacry-ransomware-unlock-files.html.

[Mag] Infosecurity Magazine. *Database Ransomware Attackers Migrate to MySQL*. URL: https://www.infosecurity-magazine.com/news/database-ransomware-attackers/.

[Mav+] Panayiotis Mavrommatis et al. *The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution Moheeb Abu Rajab*.

[Mic] Trend Micro. *New Crypto-Ransomware JIGSAW Plays Nasty Games*. URL: http://blog.trendmicro.com/trendlabs-security-intelligence/jigsaw-ransomware-plays-games-victims/.

[MNS16] Francesco Mercaldo, Vittoria Nardone, and Antonella Santone. "Ransomware Inside Out". In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, Aug. 2016, pp. 628–637. ISBN: 978-1-5090-0990-9. DOI: 10.1109/ARES.2016.35. URL: http://ieeexplore.ieee.org/document/7784627/.

[Moo16] Chris Moore. "Detecting Ransomware with Honeypot Techniques". In: *2016 Cybersecurity and Cyberforensics Conference (CCC)*. IEEE, Aug. 2016, pp. 77–81. ISBN: 978-1-5090-2657-9. DOI: 10.1109/CCC.2016.14. URL: http://ieeexplore.ieee.org/document/7600214/.

[MZL16] Monika, Pavol Zavarsky, and Dale Lindskog. "Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and Characterization". In: *Procedia Computer Science* 94 (2016), pp. 465–472. ISSN: 18770509. DOI: 10.1016/j.procs.2016.08.072. URL: http://linkinghub.elsevier.com/retrieve/pii/S1877050916318221.

[ONe] Patrick Howell O'Neill. *Ransomware demands now average about $1,000 because so many victims decide to pay up.* URL: https://www.cyberscoop.com/ransomware-demands-now-average-1077-many-people-deciding-pay/ (visited on 06/23/2017).

[Ost] Osterman. *Malwarebytes / Osterman Survey: Understanding the Depth of the Ransomware Problem in the United States.* URL: https://www.malwarebytes.com/surveys/ransomware/?aliId=13242065 (visited on 04/15/2017).

[Rog16] Ben Lelonek & Nate Rogers. *Make ETW Great Again.* 2016. URL: https://ruxcon.org.au/assets/2016/slides/ETW_16_RUXCON_NJR_no_notes.pdf.

[Sca+16] Nolen Scaife et al. "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data". In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS).* IEEE, June 2016, pp. 303–312. ISBN: 978-1-5090-1483-5. DOI: 10.1109/ICDCS.2016.46. URL: http://ieeexplore.ieee.org/document/7536529/.

[SCL15] Kevin Savage, Peter Coogan, and Hon Lau. "The evolution of ransomware". In: (2015). URL: http://www.symantec.com/content/en/us/enterprise/media/security%7B%5C_%7Dresponse/whitepapers/the-evolution-of-ransomware.pdf.

[Sco14] Katie Scoggins. "Executive Summary". In: (2014). URL: https://cyber.kent.ac.uk/Survey2.pdf.

[Sen] SentinelOne. *The Value of SentinelOne.* URL: https://go.sentinelone.com/rs/327-MNM-087/images/2016%20SentinelOne%20Statement%20of%20Value%20Brief.pdf.

[SFG] SFGate. *Hacker cost SF Muni $50,000 in lost fares, agency says.* URL: http://www.sfgate.com/bayarea/article/S-F-Muni-says-hacker-cost-agency-50-000-in-lost-10688275.php.

[Sga+16] Daniele Sgandurra et al. "Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection". In: (2016). arXiv: arXiv:1609.03020v1. URL: https://arxiv.org/pdf/1609.03020v1.pdf.

[Siz] Folder Size. *Folder Size.* URL: http://www.folder-size.com/.

[Sym] Symantec. *Can files locked by WannaCry be decrypted: A technical analysis.* URL: https://medium.com/threat-intel/wannacry-ransomware-decryption-821c7e3f0a2b.

[Sym15] Symantec. "The evolution of ransomware". In: (2015), p. 57. URL: http://www.symantec.com/content/en/us/enterprise/media/security%7B%5C_%7Dresponse/whitepapers/the-evolution-of-ransomware.pdf.

[Szy] Tomasz P. Szynalski. *What you should know about Volume Shadow Copy/System Restore in Windows 7 & Vista (FAQ)*. URL: http://blog.szynalski.com/2009/11/volume-shadow-copy-system-restore/.

[TCM] Syed Taha Ali, Dylan Clarke, and Patrick McCorry. “Bitcoin: Perils of an Unregulated Global P2P Currency”. In: (). DOI: 10.1007/978-3-319-26096-9. URL: https://link-springer-com.proxy.findit.dtu.dk/content/pdf/10.1007%7B%5C%%7D2F978-3-319-26096-9%7B%5C_%7D29.pdf.

[Tec] Ars Technica. *Online databases dropping like flies, with >10k falling to ransomware groups*. URL: https://arstechnica.com/security/2017/01/more-than-10000-online-databases-taken-hostage-by-ransomware-attackers/.

[Win] Windows. *Trojan: Win32/Procesemes.A.dll*. URL: https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Trojan%7B%5C%%7D3AWin32%7B%5C%%7D2FProcesemes.A.dll%7B%5C&%7DThreatID=-2147343025 (visited on 05/14/2017).

[Yui+04] J. Yuill et al. “Honeyfiles: deceptive files for intrusion detection”. In: *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004*. June (2004), pp. 116–122. DOI: 10.1109/IAW.2004.1437806. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1437806.