Applying machine learning to automate detection and mitigation processes for on-going ransomware attacks on Windows environments

by

Nami Shah

Dissertation submitted in partial fulfilment for the Degree of Master of Science in Cybersecurity & Management

University of Warwick

WMG

Submitted August 2018

Abstract

In 2017, roughly 60% of malware payloads were that of ransomware. With its success rate of 70%, business lost an estimated $8,500 an hour due to downtime. The global ransomware cost is believed to exceed $15 billion by the end of 2018.

The number of ransomware attacks has been actively growing since 2012, and each year they have become increasingly more potent. Although the majority of ransomware depend on being triggered by human error, the various strains and exploits available allow out-of-date security measures to be bypassed more easily.

Current solutions, such as IDS and IPS help detect and prevent attacks that attempt to exploit certain vulnerabilities. Unfortunately, these systems do not prevent human mistakes from being exploited. It has become clear that these attacks cannot simply be prevented and require adequate mitigation processes to be in place.

The need for an improved solution is self-evident and necessary. Therefore, this research focused on combining host-based monitoring and machine learning, the result of which is a model that can learn to differentiate between regular and malicious activity and recognise even the slightest changes.

This development is meant to stop a prevalent problem. Upon completion of the experiment, the model was able to detect on-going ransomware attacks within a remarkably short time-period, allowing precaution measures and mitigation processes to be activated in time to prevent any further damage.

Two variations of the model were made in order to compare overall accuracy. The first variation uses the k-nearest neighbours algorithm, whereas the second one uses the outlier detection algorithm.

A total of six live ransomware samples were deployed using *Cuckoo* in order to analyse the similarities, define a feature set for the model and collect training data using the log files

generated by enabled legacy policies. The model was trained and tested on a dataset of 1,250 log files with an 80/20 percent distribution of clean and infected event log files respectively.

The model was validated using k-fold cross-validation which resulted in an overall accuracy of 76.37% for k-NN and 74.73% for outlier detection. When tested on the same dataset which was distributed according to the 80-20 rule, k-NN resulted in an OA of 99%, whereas outlier detection was 84.31% accurate in the predictions.

Outlier detection is noticeably more stable compared to k-NN and performs significantly better on unknown threats. Although, k-NN has an undeniably better accuracy rate for known threats.

Acknowledgements

During my MSc in Cyber Security and Management at WMG, I have been introduced to many interesting professors, lecturers, and students. It is thanks to their learnings that I have gained the confidence to take up this research project.

Firstly, I would like to thank Kurt Debattista for providing me with the necessary supervision to successfully complete my research. His guidance over the year including the many meetings and discussions are very much appreciated.

Secondly, I would like to express my gratitude to Harjinder Lallie, Peter Norris and James Dorgan for peeking my interest in the more technical aspects of cybersecurity and encouraging further self-development in the field.

I would like to thank my friends for their support and believing in me, more specifically I would like to thank Danu Kumanan for the time and effort he has put into helping me understand the more mathematical aspects of machine learning.

Lastly, I would like to thank my parents and siblings for their support and motivational talks when I needed it most. They have been my moral support and anchor throughout this whole year, and that does not go unappreciated.

Declaration

I have read and understood the rules on cheating, plagiarism and appropriate referencing as outlined in my handbook and I declare that the work contained in this assignment is my own, unless otherwise acknowledged.

No substantial part of the work submitted here has also been submitted by me in other assessments for this or previous degree courses, and I acknowledge that if this has been done an appropriate reduction in the mark I might otherwise have received will be made.

Signed candidate Nami Shah

You are required to justify your submitted thesis against the degree definition for which you are registered. This needs to be downloaded and pasted into the box below.

Project definition for my degree

The dissertation for MSc Cyber Security and Management must:

1. address a research question directly relating to cyber security AND
2. demonstrate understanding of the particular issues around conducting research in the cyber domain AND
3. conduct research in the cyber domain in an appropriate manner.

In the context of the specific cyber security project undertaken, paying particular attention to the special circumstances surrounding cyber security research, one sixth of the dissertation final mark will be based on evidence of at least 15 credits (150 hours) productive activity yielding deep understanding in three of the cyber security skills groups listed below:

- Policy, Strategy, Awareness and Audit
- Legal and Regulatory Environment
- Risk Assessment and Management
- Security Architecture
- Secure Development
- Control Systems

- Information Assurance Methodologies
- Security Testing
- Secure Operations Management and Service Delivery
- Vulnerability Assessment
- Incident Management
- Forensics
- Business Continuity Planning and Management

The student will identify in their dissertation submission, which skills groups are being addressed in depth. This skills group component will receive an overall one sixth weighting when determining the final dissertation grade. The residual five sixths weighting will be derived from the generic dissertation assessment criteria.

My project relates to this definition in the following way:

This research project has consolidated previous work and research regarding ransomware analysis, detection and machine learning to develop an understanding of the missing aspects in the field. This information was used to setup a safe environment to forensically analyse the impact of ransomware on a *Windows* machine. An automation script using *PowerShell* was developed to extract and format data from this environment in preparation for the model. Furthermore, the incident response plans from two well-established organisations were compared to determine current mitigation approaches against such attacks. The entirety of this research was synthesised to develop a model that can actively detect and possibly counter the early stages of a ransomware attack. This model was developed using the Python programming language and a combination of two machine learning classification algorithms.

The following three cybersecurity skills groups are being addressed in-depth:

- Forensics
- Incident Management
- Secure Development

Table of Contents

Table of Figures

Table of Tables

Glossary

Acronym	Meaning
AD	Active Directory / Anomaly Detection
AFRL	Air Force Research Laboratory
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
APT	Advanced Persistent Threat
AV	Anti-Virus
BYOD	Bring Your Own Device
CIA	Confidentiality, Integrity, Availability
CLI	Command-Line Interface
CMU	Carnegie Mellon University
CPNI	Centre for the Protection of National Infrastructure
CPU	Central Processing Unit
CSV	Comma Separated Values
DARPA	Defence Advanced Research Projects Agency
DKIM	Domain Keys Identified Mail
DL	Deep Learning
DNS	Domain Name System
DPAPI	Data Protection Application Programming Interface
DR	Detection Rate
DT	Decision Tree
FL	Fuzzy Logic
FN	False-Negative
FP	False-Positive
GPO	Group Policy Object
HIDS	Host Intrusion Detection System
I/O	Input / Output
IC3	Internet Crime Complaint Centre
IDS	Intrusion Detection System

IOC	Indicators of Compromise
IOT	Internet of Things
IPS	Intrusion Prevention System
IPSEC	Internet Protocol Security
ITPT	Insider Threat Protection Tool
JS	JavaScript
k-NN	k-Nearest Neighbours
KVM	Kernel-based Virtual Machine
LGPO	Local Group Policy Object
MBR	Master Boot Record
ML	Machine Learning
MLP	Multilayer Perceptron
MPSSVC	Microsoft Protection Service (Service)
NB	Naïve Bayes
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
NN	Neural Network
NSA	National Security Agency
OA	Overall Accuracy
OS	Operating System
RAM	Random-Access Memory
RCA	Root Cause Analysis
RL	Reinforcement Learning
RPC	Remote Procedure Call
SAM	Security Account Manager
SIEM	Security Information Event Management
SMB	Server Message Block
SOC	Security Operations Centre
SPF	Sender Policy Framework
TF-IDF	Term Frequency – Inverse Document Frequency
TN	True-Negative

TP	True-Positive
TSB	The Shadow Brokers
UAC	User Account Control
VASA	Virtual Assistant to Security Analysts
VBS	Visual Basic Script
VM	Virtual Machine
VM	Virtual Machine
WMIC	Windows Management Instrumentation Command-Line

Table 1, Glossary

1 Introduction

This chapter begins by briefly introducing the current problem with ransomware detection in the field of cybersecurity. The second section provides some background information regarding artificial intelligence, intrusion detection systems and response plans. These are then linked together to clarify the problem further.

1.1 Motivation

With the current state of cybersecurity being unable to accurately detect and adequately prevent ransomware attacks, the need for a new and improved mitigation model is required (Slattery, 2018). With the help of artificial intelligence (AI), which has been rapidly advancing over the last few years, the ability to build an independent and smart solution has become more plausible. Unfortunately, existing AI solutions come in the form of rather expensive closed source models which requires companies to rely on third parties for maintenance and vulnerability patching.

1.2 Ransomware

Ransomware is a type of malicious software, a subset of malware, wherein access to the victim's data is blocked, usually through encryption, and a ransom is requested in order to decrypt the data and regain access. For the last couple of years, ransomware has proven to be a prominent threat to organisations of all sizes. In 2015, the Internet Crime Complaint Centre (IC3) had received over 2,453 ransomware complaints, resulting in losses of over $1.6 million (IC3, 2015). In 2016, this number increased by 200, although 50% more losses were filed (IC3, 2016).

The largest ransomware attack to date was introduced in 2017. *WannaCry* targeted computers running *Microsoft Windows* and propagated through *EternalBlue*. This exploit was released by The Shadow Brokers (TSB), a black hat hacker group well known for leaking zero-day exploits

and tools used by the National Security Agency (NSA). *WannaCry* infected over 300.000 devices and demanded up to $600 worth of bitcoins in ransom (Crowe, 2017).

Ransomware is spread through various methods, such as phishing attacks, social engineering, affiliate schemes, drive-by downloads, botnets or malvertising. Current security measures, such as Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS) and firewalls do not protect infrastructures from all malicious activity. For example, they are not able to detect and mitigate social engineering attempts, zero-day exploits, well-engineered phishing attacks or malicious code injected in legitimate software. Unfortunately, most anti-virus (AV) or monitoring software only recognise malware based on their signature. Therefore, a well-developed unique binary with an unknown signature could be deployed undetected. Additionally, certain malware operates within the memory space of other processes in order to avoid detection, or disable the taskbar from opening, e.g., *TeslaCrypt*.

In 2016, according to (Kaspersky, 2016), over sixty new ransomware families made their way into cyberspace, and the number of new strains, or ransomware modifications, increased by 11-fold. Furthermore, an individual being attacked in the third quarter of 2016 was estimated at every 10 seconds, compared to the first quarter being every 20 seconds. For businesses, the number of attacks was increased by threefold, from once every 2 minutes to once every 40 seconds. Moreover, one in five businesses who paid the ransom did not receive a decryption key (see Figure 1).

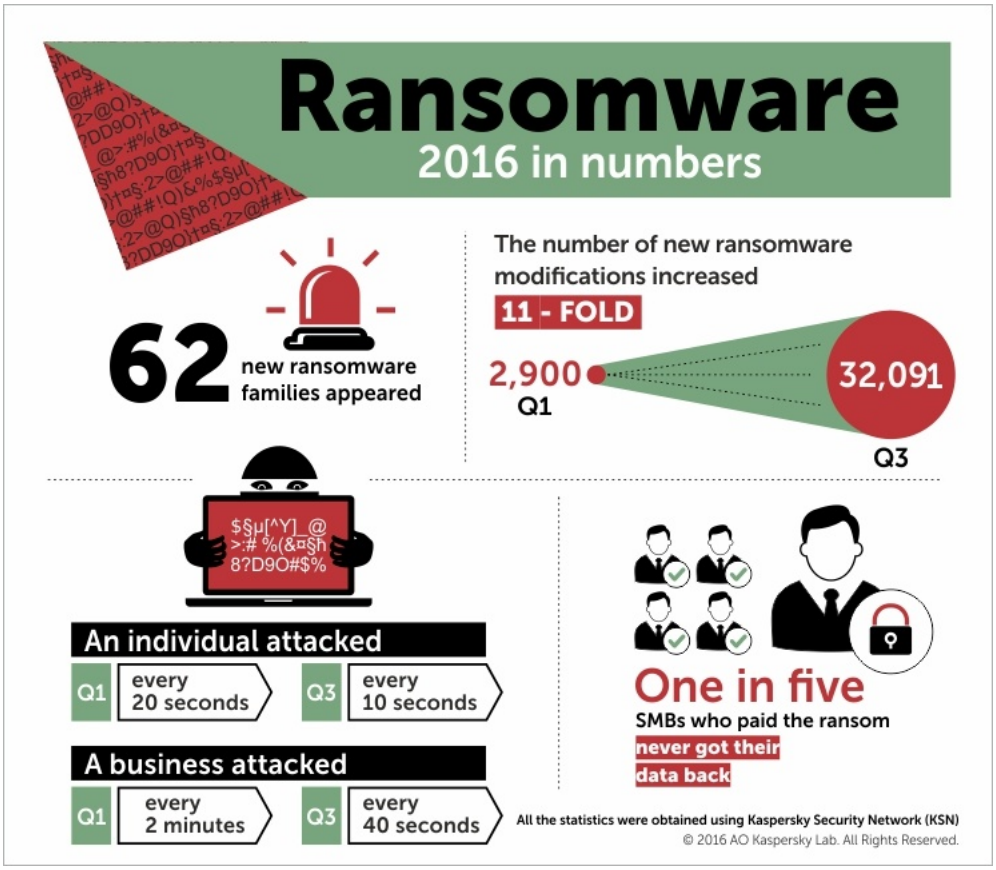

Figure 1, Ransomware 2016 in numbers, source: (Kaspersky, 2016)

Over the last several years, various new strains of well-known ransomware have appeared. Usually, some minor changes allowed them to remain undetected in order to compromise the device completely. These changes range from slowing down the encryption of files to polymorphic code or randomising the encryption processes (Nadeau, 2018).

1.3 Background

1.3.1 Artificial Intelligence

In 1956, John McCarthy held the first conference on artificial intelligence. It was there where he first coined the term AI (Smith *et al.*, 2006). Although, several years prior to that, Alan Turing published a paper on the concept of machines being able to perform intelligent actions and essentially simulate human beings (Turing, 1950).

Over the years, multiple definitions of artificial intelligence have formed, e.g., "The study of mental faculties through the use of computational models." (Charniak & McDermott, 1987), "The study of how to make computers do things at which, at the moment, people are better." (Rich & Knight, 1991) or "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages." (Oxford, 2017). AI is an umbrella term in the field of computer science and essentially means a program that can sense, reason, act and adapt based on given data.

Some of the significant problems artificial intelligence aims to solve are natural language processing (NLP), perception and learning. The latter is also known as machine learning (ML), the objective of which is to develop an algorithm that improves in performance based on the quantity and quality of data (training data) fed into the algorithm (Singh, 2016). Some of the more well-known approaches to applying these algorithms are probabilistic methods, classifiers and artificial neural networks (ANN). Neural networks can be further divided into multiple types, such as convolutional, feed-forward, recurrent, etc., each has their own advantages and disadvantages depending on the use.

As mentioned before, there are several ways to process data; these algorithms are categorised into three main categories, namely, supervised learning, unsupervised learning and reinforcement learning. Supervised learning requires the training data to be labelled and the characteristics (feature sets) to be known. This means the targets for each dataset is known. Unsupervised learning means the data is unlabelled and the algorithm defines groups based on

the similarity of characteristics it can recognise. Reinforcement learning is more goal-oriented in the sense that it uses a feedback loop and reward function to calculate the success of a performed action. This will allow the model to learn a sequence of actions in order to either improve its value function or achieve a particular goal.

Companies such as Google, Facebook, Amazon, and Apple have developed various applications that use artificial intelligence, e.g., Google DeepMind, Amazon Echo and Siri.

Specific algorithms are used depending on several aspects, such as time, processing power, dataset size, etc. Table 2 displays several of these problems as well as the algorithms that can be used to solve them.

Problem	Algorithm
Supervised Learning	
Classification	Decision trees
Regression	Ensembles
	k-NN
	Linear regression
	Naive Bayes
	Neural networks (NN)
	Logistic regression
	Perceptron
	Relevance vector machine (RVM)
	Support vector machine (SVM)
Unsupervised Learning	
Clustering	BIRCH
Reduction	CURE
	Hierarchical
	k-means
	Expectation–maximization (EM)
	DBSCAN
	OPTICS
	Mean-shift
	k-NN
	Local Outlier Factor
Reinforcement Learning	Criterion of Optimality
	Brute Force
	Value Function
	Direct Policy Search

Table 2, AI problems and algorithms, source: (dev_jamal)

1.3.1.1 Linear Regression

Linear regression aims to find the best fit line as can be seen in Figure 2. The line represents a prediction of the Y value for any given X value. The distance between the values and the regression line, also known as straight line is the margin of error. The smaller the distance, the more accurate the prediction (Von Hippel, 2015). The equation is used to minimise the error between the predicted value and the actual value. Additionally, it allows to adapt the number of parameters easily, and is not as computationally complex as compared to other algorithms, such as logistic regression. According to Von Hippel (2015) "It's only when you have a really wide range of probabilities—say .01 to .99—that the linear approximation totally breaks down.". A probability range between 0.20 and 0.80 should fit both linear and logistic models equally well. Although, linear models are easier to interpret, hence why it should be favoured.

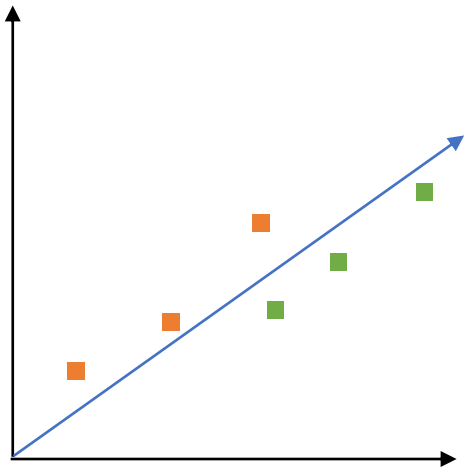

Figure 2, Linear Regression

1.3.1.2 Logistic Regression

A logistic model usually takes the shape of an S-curve as can be seen in Figure 3. The relationship between the dependant and independent variables are not necessarily linear. As a result, the margin of error is not required to be linear either. According to Von Hippel (2015) "When the true probabilities are extreme, the linear model can also yield predicted probabilities that are greater than 1 or less than 0. Those out-of-bounds predicted probabilities are the Achilles heel of the linear model.".

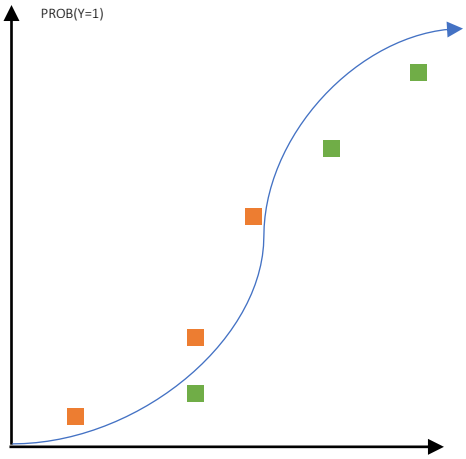

Figure 3, Logistic Regression

1.3.1.3 k-Nearest Neighbour

When there is little to no knowledge regarding the distribution of the data, the k-Nearest Neighbour (k-NN) algorithm is one of the most straightforward non-parametric classification techniques. New cases are classified based on certain similarities to previously classified cases. The **k** represents an odd value that refers to the number of nearest neighbours, which can be calculated using, e.g., a distance function. If k is equal to three, a distance function is used to calculate the three nearest neighbours where the majority vote defines the classification of the new object (see Figure 4). The value of **k** is required to be an odd value to avoid a draw in the votes. "More importantly, in a dynamic environment that requires frequent additions to the training document collection, incorporating new training documents is easy for the k-NN classifier."(Liao & Vemuri, 2002).

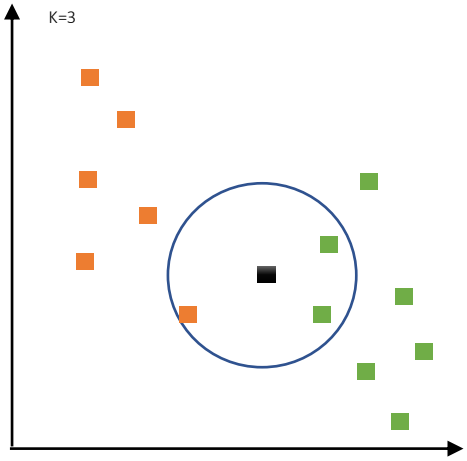

Figure 4, k-NN

1.3.1.4 Anomaly Detection

Outlier detection, or more commonly known as anomaly detection, is a type of machine learning that is quite often used on unsupervised datasets. The aim is to build a model for the probability of x, noted as $p(x)$, where x is a specific feature set of some instance. The probability threshold, epsilon, is calculated using the provided training data. Should the result of a new, unknown instance, $p(x1)$, be less than the pre-calculated epsilon, the instance is flagged as an anomaly. Gaussian distribution (normal distribution) is used in order to approximate normal behaviour, as they would be aggregated around its mean (μ) with a standard deviation of sigma (σ), which defines the spread around the mean.

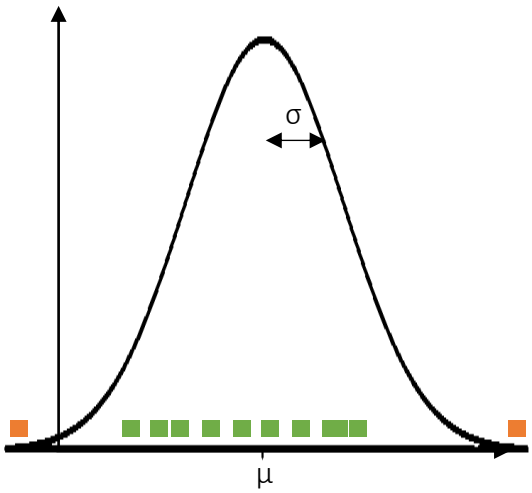

Figure 5, Outlier Detection

1.3.2 Intrusion Detection System

Depending on the type of threat one would want to detect, one or multiple intrusion detection systems can be placed either outside the firewall or within the perimeter. These systems can be either hardware or software. “In Information Security, intrusion detection is the act of detecting actions that attempt to compromise the confidentiality, integrity or availability of a resource. Intrusion detection does not, in general, include prevention of intrusions.” (Shetty & Shekokar, 2012).

(Stampar & Fertalj, 2015) provided some good insight into the high-level workings of intrusion detection systems. An IDS, either software or hardware, analyses information for suspicious activity and attempts to identify security breaches. An IDS can be either signature-based or anomaly-based. The traditional intrusion detection system is signature-based and effective for known malicious attacks. Unfortunately, they are not able to detect new variants of an attack, thus decreasing its effectiveness (Lee *et al.*, 2001). Anomaly-based IDS determines a baseline of ‘normal’ activity and detects behaviour significantly different from the baseline. Although, a high false-positive (FP) rate is expected. Network Intrusion Detection Systems (NIDS) primarily focus on network activity by intercepting requests in order to monitor for network-based threats, as opposed to Host Intrusion Detection Systems (HIDS). HIDS analyses system events for anomalous behaviour and requires the installation of an agent on each client. An IDS can be either passive or reactive based on the response. When a passive IDS is triggered, a notification is sent to the administrator or Security Operation Centre (SOC) team, who in turn will have to take action. A reactive IDS, also known as an Intrusion Prevention System (IPS), will have a sequence of pre-defined responses or mitigation actions in order to stop the threat. The most common response is IP blocking from the source.

A program that uses an algorithm to make predictions and decisions based on input data rather than being explicitly programmed to follow and execute instructions is what makes for ideal candidates for intrusion detection systems. An application of machine learning can provide companies with an alternative rather than require enormous investments for a low-efficiency signature-based solution.

A solution integrated with machine learning capabilities can learn to recognise new variants of malicious activity as well as commence mitigation procedures, such as, logging traffic and forwarding this to Security Information and Event Management (SIEM) software, notifying the SOC team, quarantining processes and blocking specific activity.

Additionally, the human factor still plays a critical role on the distribution and deployment of malware, as well as aspects that are beyond anyone's control, e.g., zero-day exploits or distributed denial of service attacks. Nowadays, it is not a matter of *if*, but *when* a cyber-attack will take place. Current security measures are implemented with the idea that a cyber-attack might take place, rather than being prepared enough to know what to do *when* it happens. This usually leads to a gap for attackers to exploit.

1.3.3 Mitigation and Response Processes

An industry report by (Ahnlab & Gartner, 2015) presented an interesting process (see Figure 6). This illustration gives a different perspective on how ransomware could be tackled before the encryption stage and represents, according to the authors, the ideal response process. A scan will traverse through the systems' file structure before the encryption phase initiates. This can be monitored and flagged, thus allowing prevention methods, such as quarantining suspicious processes and files and disconnecting from network shares, to be activated in time.

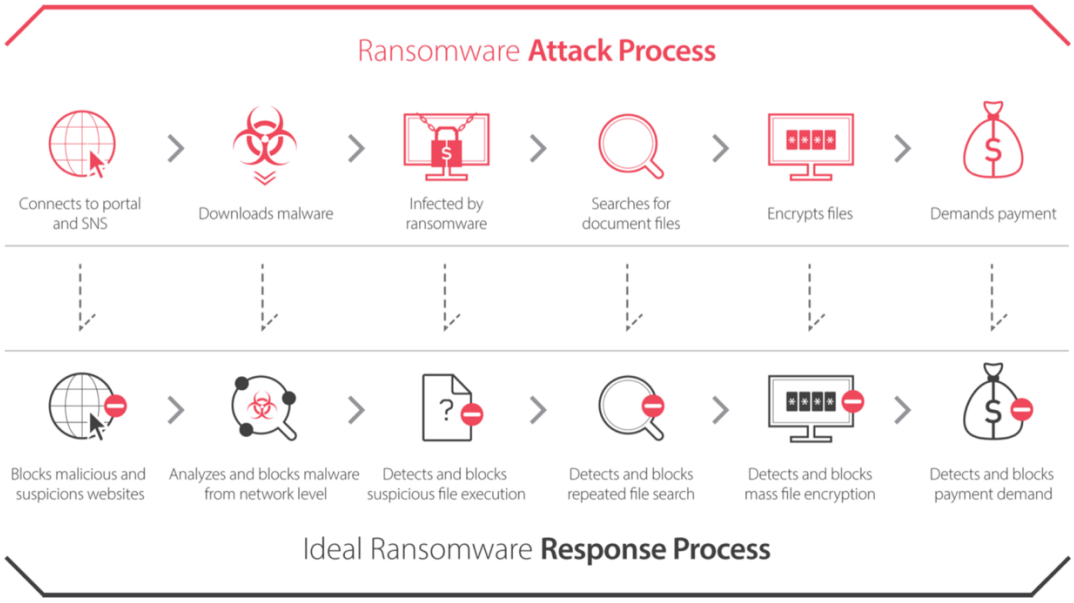

Figure 6, Ransomware Response Process, source: (Ahnlab & Gartner, 2015)

Looking into the response plan from large organisations, such as Accenture and IBM, provides a better understanding of how these situations are usually tackled.

A report by (IBM, 2016) shows how they deal with similar incidents. The incident lifecycle used and described in their report comes directly from the NIST (National Institute of Standards and Technology) security incident handling guide (Grance *et al.*, 2004) and consists of the four stages as seen in Figure 7.

Figure 7, Incident Lifecycle, source:(IBM, 2016)

"Because of the quick evolution of ransomware, IBM notes that the preparation phase of the NIST Incident Response Life Cycle is the most important. Once malicious ransomware files have been detected, it is likely too late and your files have been encrypted." IBM (2016). It is strongly suggested to use a defence-in-depth strategy in order to ensure the prevention of the environment being infected. The following actions are a few examples of what can be done in order to prepare for ransomware incidents.

- Educating end-users to become more aware and security conscious;
- Recognising ransomware events to speed up response time;
- Unexpected phishing campaigns to generate metrics;
- Increase awareness around Microsoft Office macros and the capabilities;
- Block executables from being sent through email;
- Maintain and patch current protection services;
- Block programs or scripts from being executed from the TEMP folder;
- Maintain a strict patch management policy;
- Enable DNS sinkholes, web filtering and next-gen firewalls for early detection of malicious activity;
- Set permissions to files and network shares using least privilege methodology;
- Disable Flash (contains a large number of security flaws);
- Disable Windows Scripting Host through group policy to prevent the execution of VBS and JS;
- Develop a response plan to minimise risk and damage to the assets.

Two actions which were not mentioned in the report is the physical isolation or virtual segregation of legacy devices and offsite backups. Network zoning could prevent infected legacy systems to further spread malware through network-connected devices or shares. Additionally, a typical ransomware modus operandi is the deletion or encryption of volume and shadow backups. This means infected organisations will have to, most likely, rely on offsite backups, assuming these were not virtually or physically connected to any of the infected devices or networks.

Regarding the detection and analysis phase, "The most time-sensitive issue is to identify any and all systems that have (or may have) been infected with ransomware. The reason for this is to help minimize the risk to the organization by isolating the infected systems." IBM (2016).

The report describes three scenarios in which they highlight the following actions which need to be undertaken by the user (depending on the scenario and indicators of compromise).

- Check ownership and permissions of encrypted files or newly created files to determine which host or user the encryption is happening from;
- Shutdown the device and disconnect from the network to prevent further damage;
- Notify the SOC team or IT staff immediately;
- Screen capture (using a mobile device) the display message of the ransomware to determine the strain.

The analysis phase is primarily focused on the following two aspects, namely, malware identification and root cause analysis (RCA). The former is necessary to determine the ransomware variant, as upcoming strains might have unique and unknown capabilities. The latter is necessary to determine how the ransomware got inside the organisations' environment. Both aspects are vital to preventing infection and further damage.

The containment, eradication and recovery phase use the information from the detection and analysis phase to quarantine, remove and restore the infected devices. The results of the RCA can be used to apply additional countermeasures. The following actions are suggested by the report to contain the ransomware.

- Disconnect the device from the network;
- Shutdown (ideally hibernate for forensic purposes) the device;
- Disable file sharing services;

As for eradication and recovery, it is advised to reformat the infected systems entirely and recover from backups. Finally, the post-incident activity phase includes learning from the mistakes made in order to better prepare for future attacks.

Accenture's response plan (Accenture, 2017) has several interesting points to add. For example, a program that continuously develops employees' awareness around email attacks. Additional tests are used to measure the adoption of the provided guidance. Furthermore, strengthening email controls could prevent the initial stages of malware to deploy. The following email control actions are suggested in the report:

- Activate spam filter;

- Authenticate inbound e-mail by using DomainKeys Identified Mail (DKIM);
- Prevent spoofing by using Sender Policy Framework (SPF);
- Scan emails and attachments;
- Block executable files;
- Use cloud-based email analytics tools to quarantine known threats;
- Notify users when incoming emails are originated from an external server;
- Display file extensions in the attachments;
- Use content preview applications that do not support macros.

Regarding Infrastructure protection, the report provides a more extensive list of actions. As can be seen, several similar actions are mentioned, although the focus is mainly on policy-based management and monitoring (SIEM).

- Limit administrator rights and closely monitor privileged access;
- Use endpoint protection which includes heuristic behaviour analysis and have policies in place to enforce frequent updates;
- Maintain security compliance program to validate tools and monitoring software;
- Apply strong access control lists on the network and segment server network from that of workstations;
- Have security system reviews in order to properly configure and harden firewalls, IDS and gateways;
- Deny the execution of commands by default to help keep the servers secure;
- Enforce regular patching of operating systems and applications from known vulnerabilities. Track individual assets to ensure compliance across the enterprise;
- Limit administrator access to absolutely necessary tasks;
- Configure monitoring solutions to flag incidents and enable automated clean-up methods;
- Implement web filters and URL blockers;
- Deploy cloud-based threat reputation tools to block traffic from known malicious websites (OpenDNS, Forcepoint, Palo Alto, etc.).

Note that similar actions to IBMs report were underlined. As can be seen, the remaining actions were not considered by either two companies.

To conclude, the preparation phase is the first line of defence and should not be taken lightly. Should a malicious attack get passed the security measures in place, the ideal response time would be before the encryption stage initiates. Unfortunately, considering the short time period (seconds) in which the attack is fully executed, it seems unlikely to a model could detect and mitigate the attack before this stage. Although, as seen above, there are several actions which could help an organisation better prepare for such attacks. In conclusion, a strong preparation in combination with strict mitigation procedures are critical in these situations.

1.4 Justification

Launching awareness campaigns and enforcing company policies might reduce the risk of human errors. Although, mistakes are bound to happen and not everything can be accounted for. Therefore, the primary focus should be early detection rather than prevention when it comes to dealing with the consequences of ransomware. Unusual or anomalous activity, such as a sudden increase of file property changes or calls to unknown application programming interfaces (API) could indicate early signs of an infection and trigger mitigation sequences.

Current mitigation techniques regarding ransomware include: recovering from off-site backups, relying on disaster recovery plans and even paying the ransom, which has been known not to be a reliable solution. The initial dropper (payload launcher) is quite often distributed through embedded macros in *Microsoft Office* documents. This makes it difficult to regulate with company policies as it could impact the business flow. Additionally, as mentioned before, users are the weakest link, which in the majority of the cases are exploited through either phishing or social engineering attempts. The real-time detection of ransomware allows for better and improved mitigation techniques.

The current market consists of several AI-based solutions which are specifically crafted to prevent, detect and even predict cyber-attacks. *Bitdefender* and *RANK* are two of the many companies that provide these solutions. As described by Luana Pascu in an article on *Bitdefender.com* "Detections based on machine learning algorithms are more effective than those that rely on signature-based systems, because they have high detection rates for new malware variants. When implemented in cybersecurity solutions, they can take the fight to the

next level and even detect sophisticated threats like APTs." (Pascu, 2017). *Bitdefender* has a vast variety of products that use machine learning for various purposes, such as malware and anomaly-based detection, online threats and deep learning.

According to an article on *Medium.com* (Rank Software, 2017), Virtual Assistant to Security Analysts (VASA), a product owned by *RANK,* was implemented at a leading global professional services firm in order to further inspect and analyse its features. These include real-time security monitoring, vulnerability assessment for both insider and external threats, contextualization and prioritization of vulnerabilities and user-friendly visual dashboard with the ability to further investigate vulnerabilities based on indicators of compromise (IoC). VASA was able to detect high-risk vulnerabilities, such as insecure logins as well as alert the customer about strange activity within their AWS environment. Additionally, unusual targeted DNS requests were detected, as well as access from peculiar IPs from other countries.

Although these products are highly advanced and state-of-the-art technology, they are closed source and unavailable for further development and adaptability to individual enterprise needs. Additionally, closed source solutions require a significant investment which does not always provide the flexibility a company might expect. Furthermore, the company would have a to rely on third parties for updates and patches of critical vulnerabilities in their software or system. Relying on third parties is not always as auspicious as presented. In a recent case, Russian hackers were able to use Kaspersky Labs to identify confidential and sensitive NSA documents and binaries on a previously infected machine (Corera, 2017). It later emerged that the AV had accidentally copied these documents and binaries to the Kaspersky Labs servers, thinking it was malicious, from an NSA employee's personal computer.

For the concept of AI-based models to be as accurate as possible, an extensive training dataset is required. The larger the dataset, the lesser the probability of false positives to occur. With over 80% of the global market shares of operating systems being held by *Windows*, it would be more convenient to focus on and easier to collect training data from such environments (Statista, 2018). *Windows* operating systems have a particular feature called Group Policy Objects (GPO). The concept of which is not uncommon when it comes to securing, auditing and monitoring *Windows*-based activity. It is the GPOs that enable logs to be generated within a

Windows environment. SOC teams usually use additional software such as SIEM, which allow the importation of information from various sources into one integrated framework. The logs produced by these policies can be used to examine user activity and perhaps feed the model with useful information.

An open source model allows for community-based support which can continuously improve and adapt the model. Furthermore, both the model and data can be fine-tuned to company needs, resulting in a more flexible solution and higher accuracy rate. Additionally, the current market has no similar solutions available which are able to detect ransomware using a combination of machine learning and *Windows* event logs.

1.5 Research Question and Objectives

1.5.1 Research Question

How can machine learning be applied in order to automate detection and mitigation processes of an on-going ransomware attack on a Windows environment?

1.5.2 Research Objectives

1. Research the current applications of machine learning in the context of cybersecurity and highlight the critical issues;
2. Analyse the ransomware kill chain and determine a feature set for the model;
3. Develop and evaluate an ML-based model and demonstrate the applicability of the model in a laboratory experiment;
4. Analyse the results of the experiment and conclude the key findings.

1.5.3 Research Contribution

A detailed guideline on which tools to use and how to setup and configure a safe environment to analyse and deploy ransomware samples will be presented. Additionally, this dissertation will provide an open model for enterprises to use to build their own real-time ransomware detection and prevention solution using machine learning. *Windows* event logs will be used as training data which are generated using enabled legacy GPOs. This ensures the model's adaptability to company policies.

1.6 Research Structure

The dissertation will be structured as can be seen in Figure 8.

Chapter 1 Introduction:

Chapter one introduces the current state of cybersecurity solutions for ransomware detection and provides background information regarding artificial intelligence, intrusion detection systems and modern-day mitigation methods. Additionally, it forms the base for justifying the research question and objectives.

Chapter 2 Reviewing Literature:

Chapter two reviews some of the relevant literature on ransomware detection and prevention, virtual environments and use of artificial intelligence in cybersecurity respectively. This chapter aims to complete the first research objective by fixating on the current gap and need for a better solution.

Chapter 3 Research Methodology:

Chapter three consists of the philosophical paradigm and logic behind the research approach. Furthermore, each of the objectives are justified and explained in more detail by outlining the thought process behind the experiment.

Chapter 4 Experiments Plan & Design:

Chapter four describes the setup of the environment, data collection methods, and development of the model. This chapter aims to complete the second and third research objectives by extracting the relevant information from the ransomware analysis in order to generate a feature set and collect training data for the development of the model.

Chapter 5 Experiment Results Analysis:

Chapter five aims to complete the final research objective by analysing the results from the previous chapter in order to determine the success of the experiment.

Chapter 6 Discussions, Conclusion and Future Work:

Chapter six presents the results and concludes the experiment. Furthermore, this chapter will discuss possible extensions to this research for future experiments.

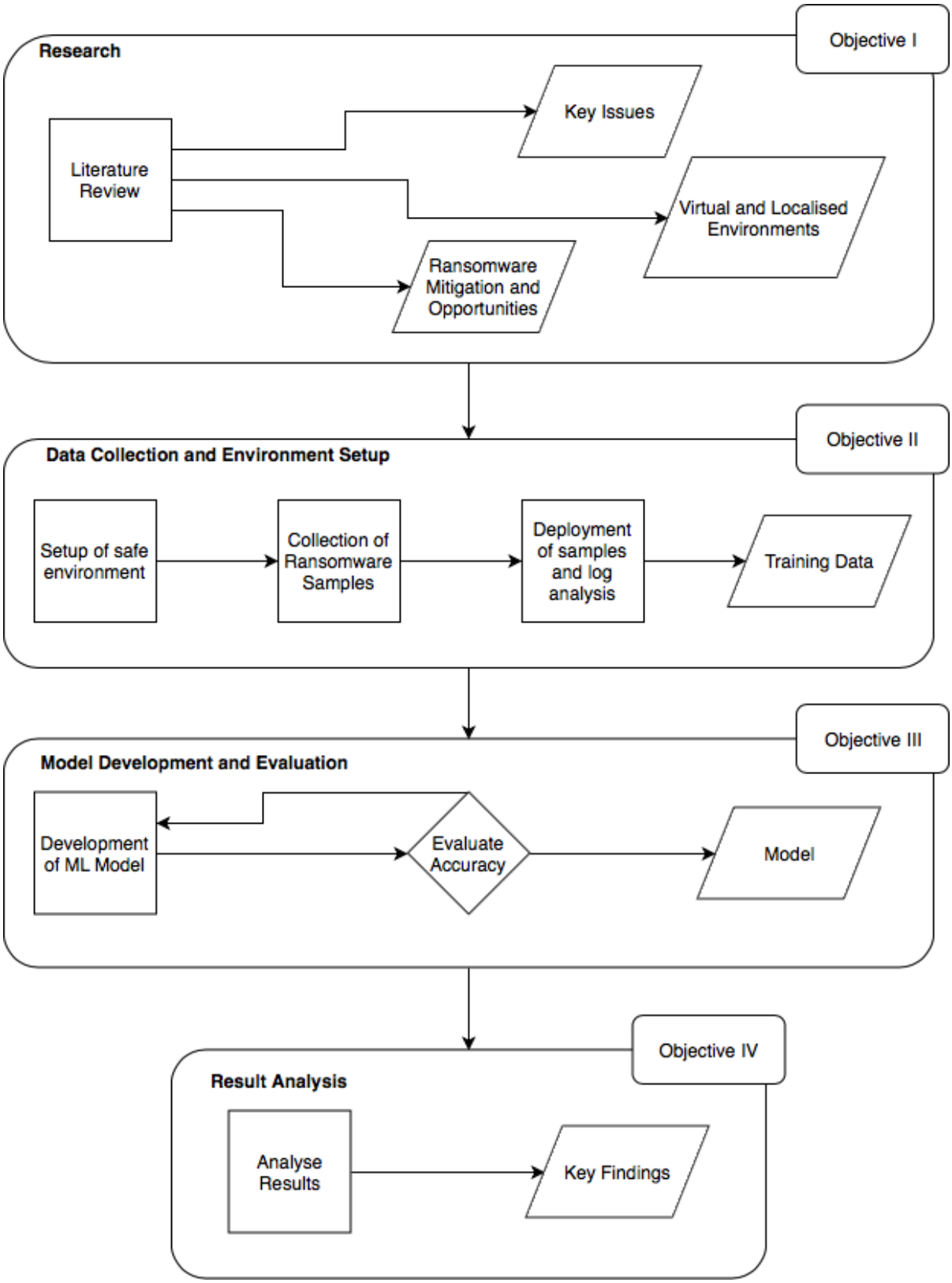

Figure 8, Dissertation Structure

2 Literature Review

This chapter contains three subsections and a conclusion. Each of the subsections aim to introduce some of the related work done in their respective fields. The first section discusses ransomware behaviour, known indicators of compromise and several detection and prevention methods and tools. The second section introduces several testing environments and techniques used in previous research in order to safely analyse malware samples. The third section focusses on the current applications of artificial intelligence in order to detect malware. Finally, the conclusion aims to highlight the learnings and clarify the research gap.

2.1 Threat Detection and Prevention

An article by (Furnell & Emm, 2017) "The ABC of Ransomware Protection" presents a brief history of various types ransomware and their impact, both financially as technically. The primary focus is on the three fundamental principles which should be taken in order to reduce the impact of a ransomware attack. The acronym ABC stands for anti-malware, back-up and critical patching. A more in-depth analysis of both the *WannaCry* and *Petya* variants are provided. They speculate that since not all NHS trusts were affected by the *WannaCry* ransomware, and each trust is responsible for their own network management and security, that the laxity of awareness and administrative management caused systems to remain unpatched. This speculation is not far-fetched since a patch was provided by Microsoft for the vulnerable *Windows* machines a month before the *WannaCry* infection broke out. Nevertheless, most systems remained unpatched until after they were forced to fix the problem. The authors concluded with possible future targets for ransomware to strike, such as Internet of Things (IoT) devices.

(National KE-CIRT/CC, 2017) analysed the *Petya* ransomware variants including *Petna*, *Petrwrap*, *NotPetya* and provided a useful summary of the vulnerable systems, prevention methods, recovery tools, IoCs, and behaviour. The advised prevention methods are similar to the ones mentioned by Steven Furnell and David Emm (2017). Although the author also advises to block the Server Message Block (SMB) protocol on port 445 TCP, disable the *Windows*

Management Instrumentation Command-Line (*WMIC*) and to avoid running systems with privileged accounts. Although, these methods are specifically directed to prevent the *Eternal Blue* exploit. SMB is a protocol used by *Windows* to transfer files on network shares. *WMIC* is a powerful command-line tool for easy administration of the machine, domain, and network. *WMIC* is similar to *PSExec*, which is a light-weight executable that allows the remote execution of processes on other machines without having to install the software manually. One of the Indicators of Compromise for *Petya* and its variants is a *PSExec* signed hash. Other indicators can be found in the registry as well as in the file system. There are two possible recovery options; if the computer was shut down before the reload, the Master Boot Record (MBR) could be re-established from the command-line interface (CLI). Occasionally, ransomware are unsuccessful at the removal of shadow copies, allowing the user to recover from a shadow back-up.

A report from (Exabeam, 2016) provides more information on the anatomy of a ransomware attack. The typical malware will attempt to remain undetected whilst executing commands. The reason for this is to buy time in order to collect credentials, exfiltrate data, potentially communicate with a Command and Control (C2) server and pivot to or compromise other systems. Ransomware payloads are easier to develop and more aggressive as the need for stealth is only required for the first couple of minutes. Once the files are encrypted, and the key has been generated, it announces its presence on the system and demands payment in return for the decryption key. After the distribution stage, the processes of ransomware run mostly autonomously, not requiring the need for any interaction from the attacker side. The authors also mention that these types of attacks are relatively new to cooperation environments, leaving security analysts clueless. After deploying over 86 strains in their own lab, the authors were able to define the six-stage ransomware kill chain depicted in the first column of Figure 9.

The cheat sheet below is provided for security analysts to know what to look after at certain stages of the kill chain in order to take swift action. Note that the common factors in all log artifacts are file activity and process tracking, which will be further discussed in §4.4.

Kill Chain Stage	Examples of Observed Ransomware Activites	Log Artifacts
Distribution	• Either an email or a web activity log showing the incoming file via either a download or an email attachment. In the most common test case, we noticed that an email with an attachment coming from a domain that had never sent an email to anyone in the company	An email Proxy Firewall IPS
Infection	• A new process on a computer is observed in process logs where the dropper is activated and read as a new process, from a new location • The process communicates to the world with a domain that can be usually identified as DGA (domain generation algorithm) artifact • A file that has been downloaded is then either renamed or ran from an unusual place such as a temp directory or a cache directory and is a new process for the user or the machine. • A process that runs another process and then dissolves is observed	Proxy File activity Process tracking
Staging	• A process that has not been seen in the enterprise before creates a randomly-named executable or an executable with the name of a Windows component but in the wrong path • A process adds an entry to an autorun location, such as the Startup folder or the Run key in the Registry • A process launches a command prompt and deletes shadow copies or modifies boot options • A newly-created process uploads data to a newly-registered domain or to a bare IP	File activity Process tracking Registry tracking Proxy Endpoint security
Scanning	• A set of authentication logs showing scanning-type activity coming from the compromised host • A process in a temporary location, or a process with a known-name but in a new location, enumerates a large number of local directories	Authentication File activity Network activity Process tracking
Encryption	• A process reads, creates, and deletes files from an unusual number of directories • Files are created with known-bad, suspicious, or unusual extensions • A process reads, creates, or deletes files in directories it doesn't normally access • A process creates many copies of a small set of files in a large number of directories	Authentication File activity Process tracking logs
Payday	• A process running from an unusual place spawns a web browser and/or Notepad with command line arguments • The desktop background is changed	File activity Process tracking

Figure 9, Ransomware Kill Chain Cheat Sheet, source: (Exabeam, 2016)

(USMAN *et al.*, 2017) focused on the *Cryptolocker* ransomware and analysed it using three specific methods, namely, surface, run-time and static code method. The surface method monitors processes such as file creation, deletion, and changes, essentially, basic observations. The run-time method observes process behaviour and impact directly related to the operating system. The static method breaks down the code into assembly instructions to determine what the ransomware actually does. Various tools were used for each of the aforementioned methods. The authors concluded that a combination of the three methods could be used to determine characteristics of ransomware and perhaps even mitigation techniques, although it is a long process.

(Magklaras & Furnell, 2001) developed a high-level architecture of an Insider-Threat Prediction Tool (ITPT). Although this is not specifically built for the detection or prevention of ransomware attacks, it can predict user misbehaviour. The tool categorises users into the following three groups:

- System masters: Legitimate system users with administrative privileges;
- Advanced users: Legitimate system users with a substantial amount of knowledge regarding the system and its vulnerabilities;
- Application users: Legitimate system users with access to necessary applications such as email, browser and database clients.

The model classifies each user into a threat profile taking into consideration the reason of misuse, such as intentional or accidental.

(Nguyen *et al.*, 2003) created a detection system based on two main factors, namely users and processes. Their system monitors file system changes and process activity to detect insider-threats. They create a profile based on what files each user usually accesses to determine anomalies, although they do not use machine learning in order to achieve this.

Both (Magklaras & Furnell, 2001; Nguyen *et al.*, 2003) built solutions that are not specifically built for the detection or prevention of ransomware attacks, nor do they use any form of artificial intelligence. Although, their primary goal is the detection of insider-threats using user

profiling methods. These methods combined with machine learning could potentially increase the detection rate (DR).

A paper by (Nieuwenhuizen, 2017) compares both static and dynamic based analysis detection methods and their limitations. For static analysis this includes its ineffectiveness against code obfuscation and targeted attacks. Furthermore, the author briefly mentions several approaches in order to extract data containing behavioural features. These include *Windows* audit logs, event tracing, kernel drivers and process hooking. Additionally, he mentions the limitations of a behavioural detection model. More advanced malware could produce noise in such a way to throw off the detection system. Where the order of calls is relevant, the malware could reorder and randomise certain calls to confuse the model. This paper does not provide a real-time, working model but explains how machine learning and behavioural activity could be combined to solve the problem.

(Kharraz *et al.*, 2016) developed a framework (UNVEIL) which automates the detection of ransomware on a large scale. Although they do not use any machine learning methods, the framework works in real-time with a zero false-positive rate. Furthermore, UNVEIL can successfully detect new families of ransomware. The UNVEIL framework is able to detect both crypto and locker ransomware by implementing a direct filesystem monitor to data buffers involved in I/O requests. These requests were analysed for attack patterns and extracted as unique fingerprints correlating to specific ransomware families. The detection of screen lockers happens through a series of screenshots and image analysis methods to detect large changes between the images.

2.2 Virtual Environment

A behavioural analysis of the *WannaCry* ransomware was performed by (Chen & Bridges, 2017) using the *Cuckoo* environment *(Cuckoo, 2010)*. *Cuckoo* is an open-source and isolated sandbox with an automated malware analysis system. Logs were collected from this environment from both regular and malicious activity. They applied a statistical information retrieval method called term frequency-inverse document frequency (TF-IDF) to determine the relevance of

certain words in the logs files. They were able to extract malware features from the generated logs automatically. Their case study was based on just one ransomware sample (*WannaCry*), but polymorphism was held into account. Unfortunately, no prevention techniques were provided, although, a good insight was given regarding generated logs and registry keys.

Krzysztof Cabaj and his team analysed network activity related to the *CryptoWall* ransomware. This was done by setting up a honeypot environment running *Maltester*, an automated run-time analytical system (Cabaj *et al.*, 2015). A honeypot server is purposely made vulnerable to various types of malware with the goal to be exploited by real black-hat hackers. The used techniques and entry points, as well as malware behaviour, is analysed in order to gain an in-depth understanding of how the hackers executed their attacks. This information can then be used to develop more advanced prevention and detection methods. The authors present the differences between static and dynamic analysis as well as the advantages. Although static analysis is safer, dynamic analysis allows the unravelling of true malware behaviour. Due to obfuscation, run-time code generation, and anti-debugging techniques, static analysis would not be able to uncover most of the activity. *TCPdump*, a network monitoring tool, was used to analyse most of the networking activity within the secured virtual machine.

(Shukla *et al.*, 2016) dynamically diagnosed three strains on physical machines rather than on virtual machines. It is known that in some cases, ransomware will deactivate or even self-destruct when launched in a virtual environment. This is usually detected by the number of processor cores used in the environment. A machine using one CPU core is more likely to be identified as a virtual machine. Therefore, only the file system was virtualised to minimise the damage using a file system filter driver, which is an optional driver that allows for file system behaviour modifications. CPU usage, memory usage, I/O calls and file properties were monitored in order to detect potential threats. A dataset of normal behaviour and activity was collected by observing over 50 users over a month time. The malicious activity dataset was collected whilst testing their solution. The variants were modified to test the solution's detection capabilities further.

2.3 Use of Artificial Intelligence in Cybersecurity

(Rieck *et al.*, 2011) created a framework that automates the analysis of malware behaviour using machine learning techniques, such as clustering and classification. By using these methods, they were able to identify up to 1,000 malicious binaries on a daily basis. The framework monitors the execution of malicious binaries using *CWSandbox*. This monitoring tool intercepts system calls using an inline function hook, which writes the calls to a log file before executing it. The behavioural patterns are embedded in a high-dimensional vector space, which is then passed to a clustering and classification algorithm. The analysis is based on common behavioural patterns found in malware, such as the modification of particular file systems or the use of specific mutexes. A mutex prevents an already infected device to be re-infected by the same malware. Although, this framework is vulnerable against possible attacks designed specifically for *CWSandbox* or forms of evasion and mimicry attacks against the analysis system.

(Narudin *et al.*, 2016) focussed on malware detection using an anomaly-based approach and classification techniques. The following four aspects were considered: basic information, content-based, time-based and connection based. As for datasets, they used *MalGenome*, which *was* a public collection of more than 1200 categorised malware. Additionally, they used a private dataset of about 30 malware as well. Unfortunately, their target platform is only focussed on Android devices.

There are a few cases where machine learning is used to automate insider-threat detection, amongst which (Legg *et al.*, 2017). Organisational log data is collected for their model to profile users based on their daily activity and job role. Each profile is observed, and a set of descriptive features is extracted on a daily basis. This is then compared to previously observed profiles in order to create a subset of anomalies. If the anomaly score surpasses a pre-defined threshold, an alert is generated. Although this system is effective, it might not be as efficient for large-scale organisations. They will require a lot of computing power in order to compare profiles of every user with previous profiles on a daily basis. Additionally, their current model does not consider organisation-dependant characteristics.

(Agrafiotis *et al.*, 2015) trained their model to detect potential threats using data collected from over 80 cases from CMU-CERT, the UK's CPNI and various news articles. Their model focusses on the detection of attack patterns/chains by clustering individual actions. The framework classifies components pertaining to insider-threats into the following four main areas, which are then used to construct and identify attack patterns:

- Attack catalyst – Motivation, trigger of the event;
- Insider-Threat characteristics – User details such as historical behaviour, work attitude, skillset, etc.;
- Attack characteristics – Attack objectives and goals;
- Organisation characteristics – Enterprise assets and vulnerabilities.

A brief but technical and in-depth experiment around artificial intelligence in network intrusion detection was done by (Stampar & Fertalj, 2015). The authors discussed previous research done in the fields of various AI algorithms to show the maturity of the subject. A total of nine AI algorithms were compared to conclude the most extensively researched one, being Neural Networks (NN), as well as the most effective one in combination with an IDS, being Naïve Bayes (NB). Both accuracy and performance were taken into consideration.

A machine learning model developed by (Soto & Zadeh, 2017) used GPOs as an active defence mechanism. After analysing common IoCs for ransomware, which includes registry changes, file attribute changes, MBR modification, removal of volume shadow files, etc., the authors trained and built a random forest with labelled data. Their model breaks down network traffic into various log files and observes for certain IoCs. The discovered IoCs are then passed to a *Python* script which automates the generation of GPOs accordingly.

2.4 Conclusion

According to Stampar and Fertalj "As faster and more effective countermeasures are required to cope with the ever-growing number of network attacks, AI comes as a natural solution to the problem." (Stampar & Fertalj, 2015).

The following characteristics distinguishes the research done in previous work from the proposed solution.

Characteristic	Description
Area	The subject matter of the research.
Result	The final product of the research.
Machine learning	The use of machine learning to achieve the result.
Event logs	The use of event logs to achieve the result.
Operating System	The operating system for which the model was developed.
Real-time	The model's capability to detect in real-time.
Data collection	The method or environment in which the data was collected.

Table 3, Literature Characteristics

Research gap:

As presented in Table 4, the research gap can be easily identified based on these characteristics. Only a few researchers have considered combining the concept of machine learning and ransomware detection to build an actual model.

The framework developed by (Rieck *et al.*, 2011) is a good example of what can be achieved with machine learning. The way they extract their input data is by intercepting system calls prior to their execution and analysing them for malicious behavioural patterns. Using this method, they would be able to prevent ransomware from executing on the machine. The model works with *CWSandbox*, an automated malware analysis environment. This makes it vulnerable against targeted attacks attempting to bypass their analysis system. The proposed model does not rely on any third-party software or dependencies except for *Python*, making it more robust.

The model by (Soto & Zadeh, 2017) tackles the problem using an active defence approach. Although, their model comes close to the proposed model, they use the random forest algorithm and train solely on network-based activity.

The model by (Narudin *et al.*, 2016) does use anomaly detection, similar to the proposed model. Additionally, they were able to collect a large dataset from *MalGenome*, a project with the aim to provide Android malware for research purposes. The predictions were made using a multidimensional vector containing a feature set from four different aspects of the malware, namely, content, time, connection and basic information. Having a feature set covering a broad range of aspects is quite effective. Unfortunately, their model is solely developed for *Android* devices.

None of the related work considered using a combination of *Windows* event logs and the k-NN or anomaly detection algorithms as the fundamental base for training and testing their models. The format of this dataset ensures the model can be adapted to company set policies as event logs are generated by enabled GPOs. Furthermore, this method could provide a more flexible, constantly-learning model with the capability of both passively and actively mitigating ransomware in real-time. Enabling host-based GPOs provides the model with training data specific to certain roles within the company. Since an event is logged for every action performed on the machine, the model would be able to detect an on-going threat in real-time. Moreover, relying on third-party environments, tools or frameworks increases risk of exploitation. Unpatched vulnerabilities might be exploited to evade or trick the analysis system. Finally, a model deployed on a domain or network level will require much more computational power than a host-based instance.

Method	Area	Result	Machine Learning	Event Logs	Operating System	Real Time	Data Collection
(Furnell & Emm, 2017)	Ransomware Prevention	Methods	—	✗	—	✗	—
(National KE-CIRT/CC, 2017)	Ransomware Prevention	Methods	—	✗	—	✗	—
(Exabeam, 2016)	Ransomware Analysis	Analysis	—	✗	—	✗	—
(USMAN *et al.*, 2017)	Ransomware Analysis	Analysis	—	✗	—	✗	—
(Magklaras & Furnell, 2001)	Insider Threats	Model	—	✗	—	✓	Network + Host
(Nguyen *et al.*, 2003)	Insider Threats	Model	—	✗	—	✓	Network + Host
(Nieuwenhuizen, 2017)	Ransomware Detection	Methods	Supervised	✗	—	✗	—
(Kharraz *et al.*, 2016)	Ransomware Detection	Model	—	✗	Windows	✓	Cuckoo
(Chen & Bridges, 2017)	Ransomware Detection	Methods	—	✗	—	✗	Cuckoo
(Cabaj *et al.*, 2015)	Ransomware Analysis	Analysis	—	✗	—	✗	Maltester
(Shukla *et al.*, 2016)	Ransomware Detection	Model	—	✗	—	✓	Host
(Rieck *et al.*, 2011)	Malware Detection	Model	Many	✗	—	✓	CWSandbox
(Narudin *et al.*, 2016)	Malware Detection	Model	Anomaly Detection	✗	Android	✗	MalGenome
(Legg *et al.*, 2017)	Insider Threats	Model	Anomaly Detection	✗	—	✓	CPNI/CMU-CERT
(Agrafiotis *et al.*, 2015)	Insider Threats	Model	—	✗	—	✗	CPNI/CMU-CERT
(Stampar & Fertalj, 2015)	Intrusion Detection	Analysis	Many	✗	—	✗	—
(Soto & Zadeh, 2017)	Ransomware Detection	Model	Random Forest	✗	Windows	✓	Network

Table 4, Research Gap Clarification

3 Research Methodology

This chapter provides more insight on the research logic and objectives. Each of the objectives are briefly explained and rationalised.

3.1 Philosophical Paradigm

A positivism research philosophy is considered as experiments usually result in scientific conclusions. Additionally, the truth is learned using factual data through observation and measurement. Moreover, an inductive approach is used as the success of the ML model will be determined by observation; thus the research is empirically observable (Saunders, 2011).

3.2 Research Logic

Although a quantitative data collection method would be considered the better option considering the positivism research philosophy, taking into account the time and resources available, a qualitative data collection method will be used instead. The experimental design method will be used which should be able to determine the relations between cause and effect and allows for more control of the situation.

3.3 Research Question

How can machine learning be applied in order to automate detection and mitigation processes of an on-going ransomware attack on a Windows environment?

3.3.1 Research Objectives

1. Research the current applications of machine learning in the context of cybersecurity and highlight the critical issues;
2. Analyse the ransomware kill chain and determine a feature set for the model;
3. Develop and evaluate an ML-based model and demonstrate the applicability of the model in a laboratory experiment;
4. Analyse the results of the experiment and conclude the key findings.

3.3.1.1 Research and Investigate Critical Issues

The first research objective aimed to highlight the current applications of machine learning and their issues in the context of cybersecurity. The result of which clarified the methods used and the gap that is caused by these closed source applications. The literature review addressed several of these issues and concluded the research gap. This was identified as a lack of the following. A *Windows*-based ML instance which does not depend on third-party environments, tools or frameworks. A model which can also be trained and tested on host-based activity generated by GPOs and actively defend by initiating mitigation procedures in a timely manner.

3.3.1.2 Analyse Ransomware and Identify Feature Set

The second research objective aims to set up a safe environment to analyse live ransomware samples, collect data and identify a feature set for the model. The details of which will be discussed in chapter 4: Experiment plan and design.

Ransomware Selection:

Regarding the choice of ransomware samples, these will be determined by the availability and types of IoCs, such as the use of registry keys for persistence, file property changes, DNS requests, and more. The samples will be taken from the following public *GitHub* repository containing a diversified collection of live malware binaries. These samples are specifically provided for educational purposes.

https://github.com/ytisf/theZoo/tree/master/malwares/Binaries.

Environment Selection:

One of the initial stages of the experiment requires the setup and configuration of an environment in order to deploy ransomware samples and collect both infected and clean training data to evaluate the model. Previous research in the field talks about setups such as *Cuckoo* and *Maltester* which allow for real-time dynamic analysis of malware and ransomware.

The *Cuckoo* sandbox was created during a Google Summer of Code project in 2010 and was part of the Honeynet Project. The open source automated malware analysis tool collects real-time data during the runtime of malware in the isolated environment. This includes the monitoring of processes and file changes. *Cuckoo* was designed to be used as a stand-alone application and can be integrated into large infrastructures. A Django web interface is available which includes database support (Honeynet, 2010).

The results of a dynamic analysis, as seen in previous work, is far more detailed compared to what one can find in a static code review. Some ransomware have the capability of detecting whether or not it is being deployed in a virtual environment, triggering an anti-debugging feature preventing the ransomware to be deployed. Although, a virtual environment does provide more safety as well as containment of malicious activity. Additionally, a physical environment will require a full reformat of the system and that of any connected devices after each deployment. Hence it would be more convenient to use a virtual environment, rather than a physical one. The anti-debugging feature will be taken into consideration during the analysis should the problem present itself.

Between static and dynamic analysis methods, the latter is the most favourable as the results are more detailed and accurate. Regarding the virtual testing environment, *Cuckoo* seems to be the most autonomous and promising solution.

Nowadays there are several platforms to setup and spawn a virtual machine (VM) while using the hardware of the original host machine. The virtual layer on top of the physical layer allows the sharing of hardware resources. This includes both volatile and non-volatile memory, CPU and GPU units, the network interface card and more. VMware *Fusion*, *Virtualbox*, *Parallels* and *Bootcamp* are several of the more commonly used virtual platforms, with each their own advantages and disadvantages.

Platform	Advantage	Disadvantage
VMware Fusion	• Very customisable • Easy to setup	• Paid
Virtualbox	• Free • Very customisable	• No integration features with the host machine
Parallels	• Easy to setup	• Paid • Limited customisation options
Bootcamp	• Separate partition on the hard drive • Better performance	• Requires reboot • No integration features with the host machine • Limited customisation options • No compatibility with other operating systems

Table 5, VM Software Comparison

Bootcamp uses a separate partition on the hard drive and requires a reboot in order to launch. This impacts the performance of the machine significantly. Unfortunately, *Bootcamp* is an *OS X* software and does not support any other operating systems; furthermore, it does not allow for much customisation. *VMWare Fusion* and *Parallels* provide an easy installation and setup guide, making it more user-friendly. Furthermore, *VMWare Fusion*, as well as *Virtualbox*, provide a highly customisable environment. Unlike *Virtualbox*, both *Parallels* and *VMware Fusion* are paid platforms, making *Virtualbox* the logical choice.

Table 6 shows the percentage of *Windows* versions used worldwide over the last year, starting from March 2017. As mentioned above, the Event Log API was only included in the *Windows* OS starting from the Vista version, which makes this feature available for almost 96% of *Windows* machines worldwide.

Windows OS	Usage in the last 12 months
Windows 10	43.92%
Windows 8.1	7.98%
Windows 8	2.43%
Windows 7	41.57%
Windows Vista	0.61%
Windows XP	3.42%
Other	0.07%

Table 6, Usage Windows OS, source: (Statcounter, 2018)

The *Cuckoo* guest virtual machine will be setup with a *Windows 10* image and will be given 4GB RAM, double the amount as the recommended minimum. *Windows Firewall*, *Windows Defender* and UAC (User Account Control) will be disabled to allow a complete analysis without disruption. UAC aims to improve security by limiting certain actions and requiring administrative authority.

Data Selection:

Group policy objects can be defined in a *Windows* environment to restrict, prevent and log certain actions. These are mostly used by system administrators to secure against and monitor malicious activity. Essentially, group policies ease the management and configuration of services, applications, and settings in an Active Directory (AD). Although, a group policy, also known as local group policy (LGPO) can be applied on non-domain, stand-alone computers (Microsoft, 2008b).

The *Windows* Event Log feature is included in *Windows* operating systems (OS) starting from *Windows Vista* and *Windows Server 2008* and supersedes the Event Logging API. Event consumers, such as Event Viewer, can use this feature to read and render captured events. The *Windows* Event Log feature provides a centralised, single collection of logs generated by both hardware and software (Microsoft, 2011). Specific events can be triggered by GPOs, making it easier to monitor malicious attempts or dubious actions.

For example, file property changes and importing *PowerShell* encryption modules can trigger an informational log, which the model can use to predict whether or not a ransomware attack is currently on-going. The information in Table 7 was directly acquired from (Microsoft, 2008a).

Event ID Range	Description
4000-4007	Group Policy start events: These informational events appear in the event log when an instance of Group Policy processing begins.
4016-4299	Component start events: These informational events appear in the event log when a component of Group Policy processing begins the task described in the event.
5000-5299	Component success events: These informational events appear in the event log when a component of Group Policy processing successfully completes the task described in the event.
5300-5999	Informative events: These informational events appear in the event log during the entire instance of Group Policy processing and provide additional information about the current instance.
6000-6007	Group Policy warning events: These warning events appear in the event log when an instance of Group Policy processing completes with errors.
6017-6299	Component warning events: These warning events appear in the event log when a component of Group Policy processing completes the task described in the event with errors.
6300-6999	Informative warning events: These warning events appear in the event log to provide additional information about possible error conditions with the action described in the event.
7000-7007	Group Policy error events: These error events appear in the event log when the instance of Group Policy processing does not complete.
7017-7299	Component error events: These error events appear in the event log when a component of Group Policy processing does not complete the task described in the event.
7300-7999	Informative error events: These error events appear in the event log to provide additional information about the error condition with the action described in the event.
8000-8007	Group Policy success events: These informational events appear in the event log when the instance of Group Policy completes successfully.

Table 7, Event ID Descriptions, source: (Microsoft, 2008a)

Feature Selection:

Event log information will be extracted after the deployment of a set of live ransomware samples. Additionally, the same will be done after the execution of several random regular actions, such as renaming or compressing files. These log files will be analysed and compared to common IoCs, the result of which will identify a format and feature set for the model, e.g., a group of high-risk event IDs. For the proof of concept, the model will be trained on both infected and clean event log information. Although, the generation and extraction of clean data can be automated, infected data will have to be extracted manually due to the encryption process. Data augmentation is foreseen for infected log information due to the low number of live ransomware samples available. A script will be developed to automate the extraction of these logs. The result of which will be a comma-separated value (CSV) file with two values, the event ID and the occurrence of that event in a time interval of 30 seconds. This strategy was chosen to reproduce a realistic approach when deploying the model in an organisation.

3.3.1.3 Develop and Evaluate Model

The fourth research objective aims to develop an open-source machine learning model with the ability to detect and initiate mitigation processes. The model will be written in *Python* as it provides various libraries to ease the implementation of such models.

Machine Learning Task Selection:

Current machine learning applications and use cases were researched in order to clarify the issues and develop a solution. During this research, the following machine learning tasks were considered: supervised, unsupervised and reinforcement learning. Supervised learning algorithms result in a predictive model and requires the training data to be labelled. Usually, supervised learning is used to solve classification problems. Unsupervised learning algorithms provide more of a descriptive model and are more commonly used in clustering problems. Lastly, reinforcement learning (RL) algorithms can determine behaviour within a specific context. This type of learning is more commonly used in robots and machines and relies on a reward function with many output possibilities (Fumo, 2017). RL actively trains in a production environment by essentially mutating certain actions and learning from the feedback function, making it unfit for this specific experiment.

Supervised learning is the obvious choice as the training data will be labelled and the model is expected to predict whether or not the current state of a machine is infected. The ransomware kill chain (Figure 9) was observed to be consistent according to a report by Exabeam (2016). As a result, mitigation procedures will essentially remain similar for most ransomware attacks. Therefore, a descriptive unsupervised model, e.g. knowing the ransomware type, is not considered for this experiment. Within supervised and unsupervised learning, deep learning (DL) algorithms can learn the various representations of the data, which could correspond to the machine being either infection or in a clean state. Deep learning is a data-intensive model where pattern recognition and correlations improve significantly according to the quantity of data it is fed. This means that DL requires more computational power, hardware, and training data, whereas other algorithms do not (Bhatia, 2017), which would be more appealing for companies.

Algorithm Selection:

The following two supervised classifier algorithms will be implemented: k-NN and anomaly detection. Both algorithms are relatively easy to implement and do not require an extensive dataset compared to, e.g. neural networks. The k-NN algorithm was chosen for its simplicity and natural handling of multi-class cases. Anomaly detection was chosen for its realistic dataset requirement, as it would rely on a higher amount of one type of class, e.g. clean data. This seemed to be a more realistic approach when implementing the model in an actual organisation.

Evaluation Methods:

The evaluation will be based on mainly two factors, namely, precision and recall. This includes factors such as false-positives (FP), false-negatives (FN), true-positives (TP), true-negatives (TN). Two validation methods will be used, one of which will split the data in an 80-20 percent ratio used for training and testing respectively. The second method will validate the performance using k-fold cross-validation, essentially rendering unseen data for testing for each ransomware sample individually. This approach was chosen to compare results between ransomware threats which the model has seen a variation of, compared to ransomware threats which the model has never seen before.

3.3.1.4 Analyse and Conclude Findings

The final research objective aims to analyse the results of the experiment to determine whether or not the model results in a more flexible and reliable solution compared to existing ones. The results will be consolidated into a conclusion which could help and perhaps even accelerate the process for future research.

3.3.2 Research Structure

Figure 10 shows the structure in which each of the chapters and objectives pass along information to the next one.

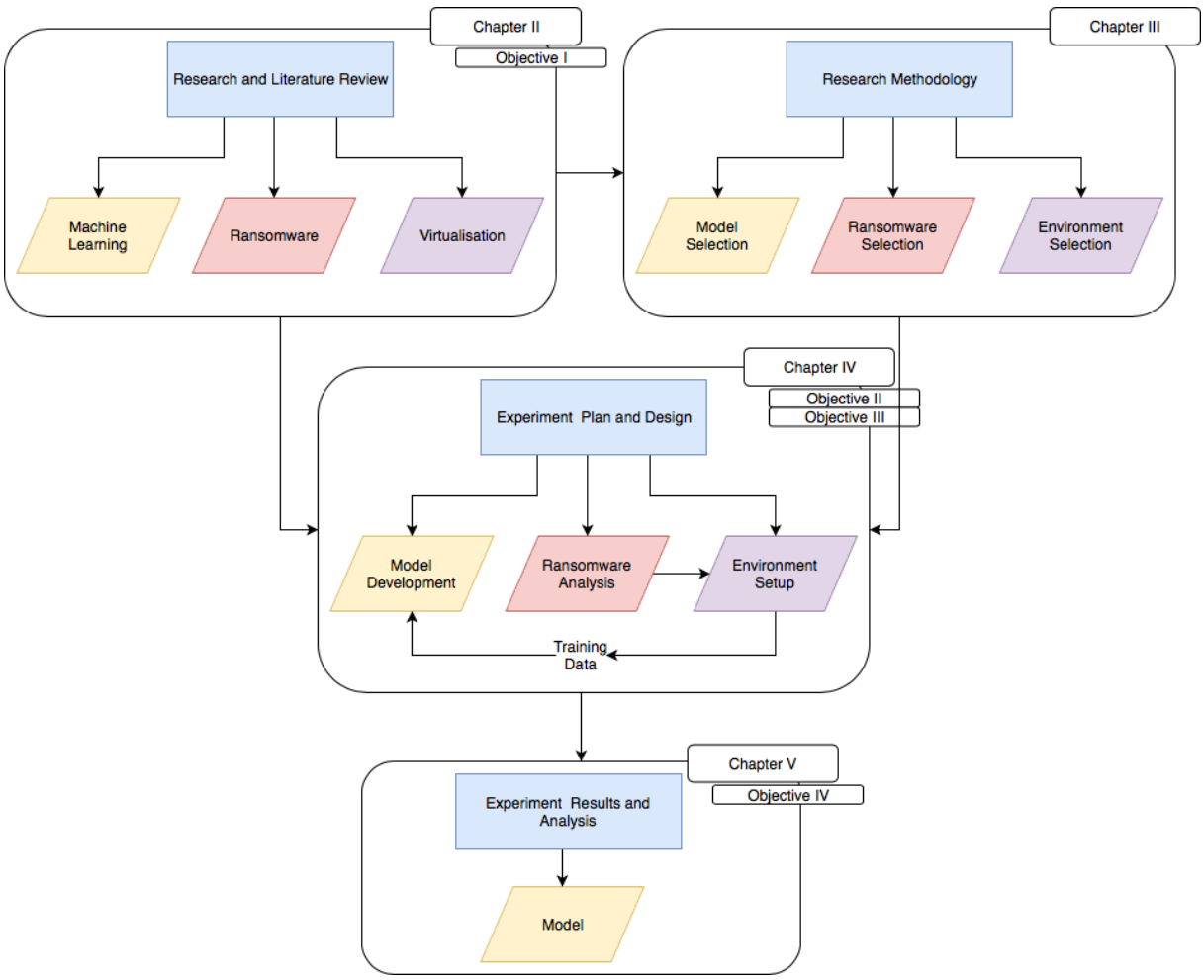

Figure 10, Research Structure

3.4 Summary

Based on the literature review, a research gap was identified. The research methodology introduces the logic behind the experiment in the next chapter. Additionally, it provides the experiment with the following structure. Firstly, a selection of ransomware samples will be made based on factors, such as availability, attack vector, impact, etc. Secondly, the environment will be chosen and setup in order to deploy the selected ransomware samples. Thirdly, data will be collected, and the damage will be analysed in preparation for the model. Next, the machine learning algorithms will be chosen based on factors, such as ease of implementation and dataset size requirements. Finally, the model will be developed and evaluated using the collected data. The results of which will be analysed to determine the success of the model.

4 Experiment Plan and Design

This chapter consists of six sections, namely ransomware selection, environment setup, ransomware analysis, feature extraction, data collection and model development. The first section reasons the choice of ransomware samples which will be deployed for analysis in the virtual environment. The second section provides an in-depth description of how the virtual environment will be setup. The third section provides the results of the *Cuckoo* analysis which is used for the feature extraction and data collection. The final section uses this information to develop the actual model using the algorithms discussed in the research methodology (§3.3.1.3).

4.1 Ransomware Selection

According to (Savage *et al.*, 2015), there are two main types of ransomware as seen in Figure 11.

Figure 11, Ransomware Types, source:(Savage et al., 2015)

The locker ransomware disables specific interface features, leaving the user with a limited ability to perform tasks, usually restricted to only interact with the ransomware. As it usually does not affect any of the user files, and can be removed, it is not as effective as the more common crypto ransomware. Hence the experiment will be primarily focused on crypto ransomware. The selection of ransomware will be based on the availability of live samples in combination with the attack vector and popularity of the ransomware.

Table 8 represents several of the top ransomware attacks in 2017. Note that the Petya variant NotPetya is not considered an actual ransomware, as it permanently overwrites the MBR. This does not allow for the data to be recovered, thus classifying it as a wiper, as it technically does not hold anything ransom. Although, it does trick the user into believing it is a ransomware by demanding a fee in exchange for a decryption key.

#	Name	Type	Ransom	Attack Vector / Characteristics
1	NotPetya	Wiper	0.99 BTC	Exploits DHCP
2	WannaCry	Crypto	$300 (BTC)	Exploits SMB
3	Locky	Crypto	1 BTC	Phishing Emails
4	CrySis	Crypto	$600 (BTC)	Exploits RDP
5	Nemucod	Crypto	0.52 BTC	Phishing Emails
6	Jaff	Crypto	0.356 BTC	Phishing Emails
7	Spora	Crypto	/	Google Chrome
8	Cerber	Crypto	1.24 BTC	Phishing Emails, RAAS
9	Cryptonix	Crypto	/	/
10	Jigsaw	Crypto	$40 (BTC)	Phishing Emails
-	Teslacrypt	Crypto	2 BTC	Malvertising, Phishing Emails
-	Vipasana	Crypto	/	Public key built-in, no internet required
-	Zerolocker	Crypto	$300 (BTC)	Phishing Email, drive-by downloads

Table 8, Ransomware Comparison, source:(Savage et al., 2015; Teplow, 2017)

Most of these ransomware samples are not currently active apart from *WannaCry*, *Cerber*, and *Jigsaw*. Hence the following alternative samples were included as well: *Teslacrypt*, *Vipasana*, and *Zerolocker*. Considering the impact each of these ransomware had and the variation in

attack vectors, they make for ideal candidates to validate the model using k-fold cross-validation.

4.2 Environment Setup

As mentioned in §3, *Cuckoo* will be used as a sandbox environment. The *Cuckoo* documentation (Cuckoo, 2010) provides a guideline for the setup of both the host and guest machines. The host consists of the machine that will run the sandbox environment. The guest consists of the machine that will run within the sandbox environment. The ransomware samples will be deployed within the guest machine. The *Cuckoo* main architecture is setup as seen in Figure 12. The setup will be slightly different as there is no need for multiple guest machines.

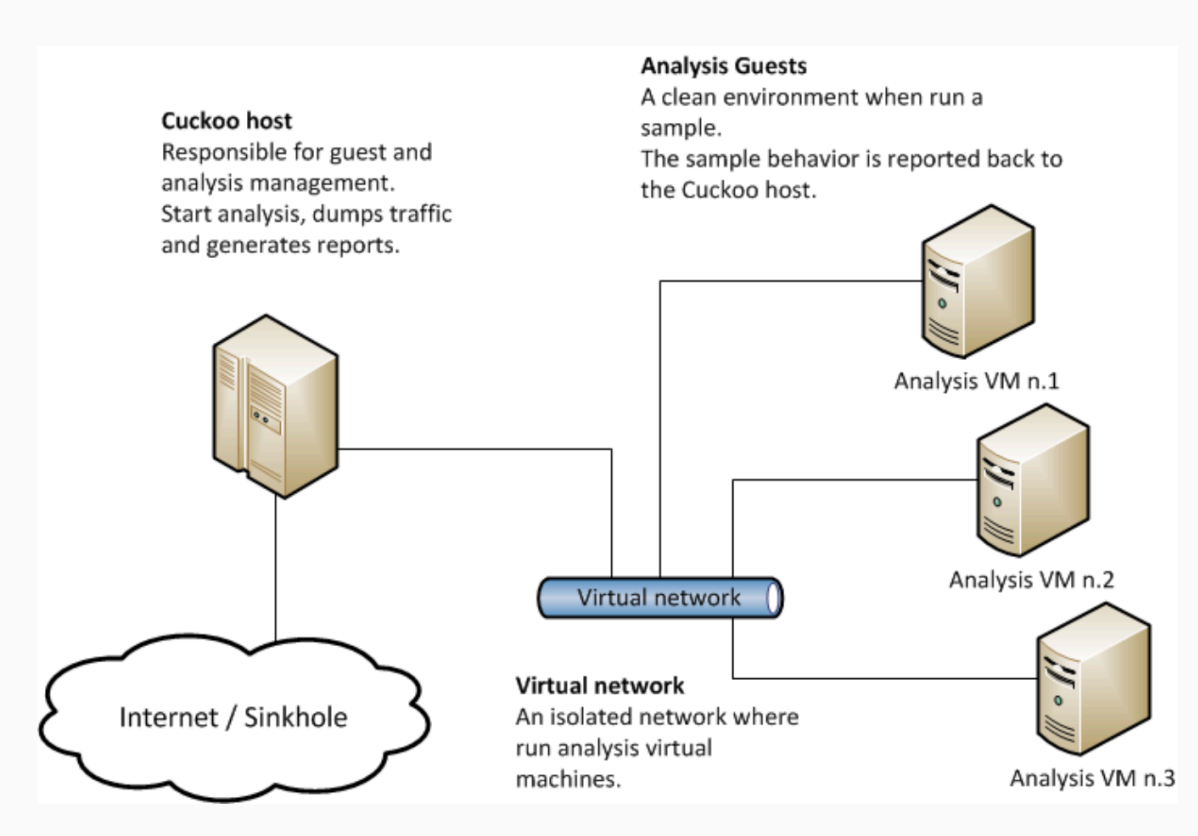

Figure 12, Cuckoo Architecture, source: (Cuckoo, 2010)

4.2.1 Host Machine

The *Cuckoo* host will be running the latest version of Ubuntu (18.04 at the time of writing), as recommended by the *Cuckoo* documentation. Additionally, *Cuckoo* 2.0.6 will be used for the analysis. The configuration of the host system can be seen in Table 9.

Memory	15.6GiB
Processor	Intel® Core™ i7-4960HQ CPU @ 2.60GHz × 8
Graphics	NVE7
GNOME	3.28.1
OS	Ubuntu 18.04 LTS 64-bit
Disk	245.1 GiB

Table 9, Cuckoo Host Configuration

The hardware of the host machine is that of a MacBook Pro late 2013 model. Approximately 250GB was partitioned to FAT32 on an external hard drive for the setup of the ext4 file system. The bootloader (Grub) was installed on the original MacBook Pro HD partition, allowing for dual boot.

The installation of *Ubuntu* was done through a live install using from a USB device onto the external hard drive partition. The timezone was set to London and a standard installation was executed. Once booted in Ubuntu, an ethernet cable was used for Internet access to install the necessary drivers and enable the Broadcom Wi-Fi adapter. The commands listed in Appendix A under subsection [1] were executed to update the system and enable networking tools (such as *ifconfig*).

Below this point, the *Cuckoo* documentation section on *preparing the host* was used to install the necessary dependencies and requirements for the host environment.

The host machine is responsible for the management of the attached guest machines and the analysis of the deployed ransomware, as well as for traffic dumping and report generation. The

commands listed in Appendix A under subsection [2] were executed for the installation of *Python* libraries, this includes the *virtualenv* library, which will be used for the isolation of the *Python* environment.

The commands listed in Appendix A under subsection [3] were executed for the installation of *MongoDB*, for the *Django* web interface and *PostgreSQL*, the recommended database.

At the time of writing, *Cuckoo* is only compatible with *Virtualbox* versions 4.3, 5.0 and 5.1. Hence why *Virtualbox* 5.1 will be installed. The commands listed in Appendix A under subsection [4] were executed in order to do so.

The following tools are crucial for a complete analysis: *TCPdump*, *volatility*, *M2Crypto* and *guacd*. The installation of a network sniffer is required for the capturing and dumping network activity caused by the ransomware. *AppArmor*, a *Linux* security module, needs to be disabled for the creation of the pcap files (network traffic dumps). The commands listed in Appendix A under subsection [5] were executed in order to achieve this.

Understandably, *TCPdump* will require root privileges. By executing the commands listed in Appendix A under subsection [6], it would be possible to allow specific capabilities to that binary without having to run *Cuckoo* as root. The result can be verified using the *getcap* command.

Although *Volatility* is an optional tool, it does give more insight as it allows for the analysis of memory dumps. Any *Volatility* version above 2.5 is recommended. The following documentation was used for the installation of *Volatility*:

https://github.com/volatilityfoundation/volatility/wiki/Installation.

The installation and setup of *Volatility* and the *distorm3* dependency were done using the commands listed in Appendix A under subsection [7].

The installation of *M2Crypto* tool was done using the commands listed in Appendix A under subsection [8]. The *OpenSSL Python* wrapper allows for the implementation of SSL in clients and servers.

The installation of *guacd* was done using the commands listed in Appendix A under subsection [9]. This service allows for an additional layer which translates RDP, VNC and SSH protocol functionality over remote communication using the *Cuckoo* web interface.

The commands listed in Appendix A under subsection [10] were used to add a user specifically to the *Cuckoo* user group. The commands listed in Appendix A under subsection [11] were used to install *Cuckoo* and several of the dependencies. *Cuckoo* can be setup in a virtual environment (virtualenv), which isolates the usage of Cuckoo, thus making it more secure, rather than being accessible globally. This can be done using the commands listed in Appendix A under subsection [12].

The current users of this system can be seen in Table 10.

Privileges	**Username**	**Password**
Administrator	NS	ubuntu
User	Cuckoo	cuckoo

Table 10, Host Users

Configuring the network slightly side-tracks from the original documentation. A default *Windows 10* VM was setup using Virtualbox according to the configurations mentioned in Table 11. A host-only network adapter was setup with the name 'vboxnet0' and IP range 192.168.56.0/24. As the current network is using a WI-FI adapter rather than an ethernet cable, some of the configurations mentioning 'eth0' were replaced with 'wlp3s0' (according to the new name scheme). The iptables rules listed in Appendix A under subsection [13] were added to enable communication between the host and guest machines; additionally, packet forwarding was enabled to allow the exchange of data between guest and host machines.

As iptables reset after every reboot, the commands were added to a script, which is executed at boot. The setup of a tap network interface in order to avoid having to start them as root can be done as presented in Appendix A under subsection [14].

The following broadcast interface was then added to the `/etc/network/interfaces` file.

```
1. auto br0
2. iface br0 inet dhcp
```

Next, the commands listed in Appendix A under subsection [15] were executed to setup the broadcast interface and a tap network for the guest machine to communicate with the host. Note that the interface name is set to the guest machine name preceded by 'tap_'. The tap interface works as a wiretap and allows a full analysis of packets destined to and sourced from the guest machine.

4.2.2 Guest Machine

The configuration of the guest system can be seen in Table 11.

Memory	4.00GiB
Processor	Intel® Core™ i7-4960HQ CPU @ 2.60GHz
OS	Windows 10 version 1703 Education 64-bit
Disk	50.0 GiB
Local IP	192.168.56.101

Table 11, Cuckoo Guest Configuration

The guest machine will be used to deploy the malicious samples and report back to the host machine using a *Python*-based API.

The latest (at the time of writing) updates were installed including KB2267602, KB890830, KB4284874, KB2267602, etc. Additionally, for the *Cuckoo* analysis to fully work, the following actions were taken:

- *Python* 2.7 was installed;
- UAC was disabled;
- The firewall was disabled;
- *Windows Defender* was disabled;
- The .NET framework 3.5 was enabled.

In order to allow further manual analysis should it be deemed necessary; the standard nine legacy policies were enabled as well. This is discussed in more detailed in §4.4. The current users of this system can be seen in Table 12.

Privileges	**Username**	**Password**
Administrator	Nami	windows

Table 12, Guest Users

Furthermore, the agent.py script was deployed on the guest machine with administrative privileges. The agent works as an API handler, managing the requests and responses coming in from the host machine. Moreover, several files were placed in the `downloads`, `desktop` and `recycle bin` folders, in order to resemble a realistic environment. These files will also confirm a successful ransomware attack. As recommended by the documentation, a snapshot 'baseline' was taken of the state of the machine, which *Cuckoo* will automatically start from and recover to after each analysis.

4.2.3 Cuckoo Deployment

Cuckoo and the web interface can be deployed using the commands below. The web interface makes it easier to submit ransomware samples and customise specific features such as memory dumps and time-outs.

```
1. $ cuckoo
2. $ cuckoo web
```

After the submission of the ransomware samples through the web interface. The timeout option is increased to 300 and the *Volatility* memory dump is enabled followed by the initialisation of the analysis. Once the analysis is done, a report is generated which can be viewed in the web interface as well. Moreover, the relevant files are stored in `~/.cuckoo/storage/analysis/<TASK ID>`.

4.3 Ransomware Analysis

The *Cuckoo* analysis generated a `report.json` file which contains a summary of most of the activity. Appendix AC.1 represents a partial output of the report file generated after deploying *Teslacrypt*. Several processes were called, under which, `lsass.exe` (local security authority subsystem service) and `oojdsu.exe`, the *Teslacrypt* payload. `Lsass.exe` is used to enforce security policies on a *Windows* system and is located in the `system32` folder. This binary is often misused by malware and can be considered fake should it be running from any other location. Furthermore, we can see `oojdsu.exe` loaded and used several DLL files necessary to perform, e.g., encryption and networking services. Finally, a registry key is added in order for the ransomware to remain persistent.

The majority of the reports consists of process calls made by each of the deployed ransomware samples. These calls are categorised as can be seen in Appendix AC.2. Table 13 represents the number of calls per category per sample. Unfortunately, *Cuckoo* was unable to automate the deployment of *Vipasana*, *Zerolocker* and *Jigsaw* due to the requirement of further manual interaction with the binary. Therefore, these were manually deployed, and similar activity was observed. Although, it is quite clear that the majority of calls come from system, process, registry and file actions. Further manual analysis and observation of the infected machine using event viewer confirmed this.

Ransomware	#System	#Process	#Registry	#File	#Network	#Crypto	#Synch	#Misc
WannaCry	16,842	1,958	7,955	43,317	2	848	3,866	941
Cerber	6,264	2,750	12,785	3,877	2,575	317	600	207
Teslacrypt	56,112	54,795	1,287	37,177	175	244	493	18

Table 13, Process Calls Comparison

The event viewer groups certain event logs together according to the following categories:

- Application
- Security
- System
- Setup
- Forwarded Events

After manually analysing the event log files in the guest machine after the deployment of a sample, the first three categories seemed to be collecting most of the events, which corresponds with the *Cuckoo* report. The last two categories did not collect any events. Hence, the custom filter for the data collection script can be defined as seen in Appendix B.

4.4 Data Collection

4.4.1 Environment Setup

The same *Cuckoo* guest environment can be used for the collection of training data. Several additional configurations need to be applied in order to do so.

The *Cuckoo* analysis determined some the behavioural aspects of ransomware, which can be further analysed in order to enable only certain policies on the guest machine. There are nine legacy auditing policies which can be enabled locally. When working with a domain, these should be enabled on a domain level instead.

Using the group policy management editor tool, the following local audit policies can be found in the directory: `Windows Settings/Security Settings/Local Policies`.

The descriptions provided in Table 14 were directly acquired from (Smith, 2008).

#	Policy	Description
1	Audit Account Logon Events	Credential Validation
		Kerberos Authentication Service
		Kerberos Service Ticket Operations
		Other Account Logon Events
2	Audit Account Management	User Account Management
		Computer Account Management
		Security Group Management
		Distribution Group Management
		Application Group Management
		Other Account Management Events
3	Audit Directory Service Access	Directory Service Access
		Directory Service Changes
		Directory Service Replication
		Detailed Directory Services Replication
4	Audit Logon Events	Logon
		Logoff
		Account Lockout
		IPsec Main Mode
		IPsec Quick Mode
		IPsec Extended Mode
		Special Logon
		Other Logon/Logoff Events
		Network Policy Server
5	Audit Object Access	File System
		Registry
		Kernel Object
		SAM
		Certification Services
		Application Generated

		Handle Manipulation
		File Share
		Filtering Platform Packet Drop
		Filtering Platform Connection
		Other Object Access Events
6	Audit Policy Change	Audit Policy Change
		Authentication Policy Change
		Authorization Policy Change
		MPSSVC Rule Level Policy Change
		Filtering Platform Policy Change
		Other Policy Change Events
7	Audit Privilege Use	Sensitive Privilege Use
		Non-Sensitive Privilege Use
		Other Privilege Use Events
8	Audit Process Tracking	Process Creation
		Process Termination
		DPAPI Activity
		RPC Events
9	Audit System Events	Security State Change
		Security System Extension
		Security Integrity Events
		IPsec Driver Events
		Other System Events

Table 14, Legacy Audit Policies, source: (Smith, 2008)

Looking at Table 14, in combination with the results of the *Cuckoo* analysis, it is easier to determine which policies to enable for the right amount of information to be logged in the event viewer, yet limiting noise, increasing overall accuracy. This heavily depends on the type of organisation and activity. For the experiment, all the legacy policies were enabled except Audit Policy Change, as this was not observed during the analysis.

Before initiating the data collection process, the following command was executed in a privileged *PowerShell* prompt to clear the current event logs to ensure all the generated events are the result of malicious activity.

```
1. Set-ExecutionPolicy unrestricted
2. wevtutil el | Foreach-Object {wevtutil cl "$_"}
```

Prior to taking the snapshot 'baseline_data_collection', the ransomware samples were downloaded and prepared for deployment. Once deployed, the custom filter results can be exported to a CSV file (comma separated values). For infected log files, this will be done manually as automation scripts will be encrypted as well. For clean log files, this will be done using a *PowerShell* script as can be seen in Appendix D. Additionally, a random action *PowerShell* script (Appendix E) perform random actions in order to generate realistic log files for a clean environment. The actions range from creating files and folders to initialising applications and browsing activity. To allow a larger dataset to be collected, the script will automatically extract log files every 30 seconds. Due to the lack of live ransomware samples available, the number of infected log files will be increased using a data augmentation script (Appendix F).

A total of 1,250 logs will be collected of which 1,000 will be clean and 250 will be from infected machines, resulting in an 80-20 percent distribution (see Table 15). I believe this to be ideal for training and testing purposes as it reflects a realistic approach. The k-fold cross-validation method will be applied to test the model's performance (Figure 13). This is done by dividing the complete dataset into two parts for testing and training. The testing data will only contain infected and clean samples which the model has not been trained on. This is iterated six times for each ransomware sample (see Table 17).

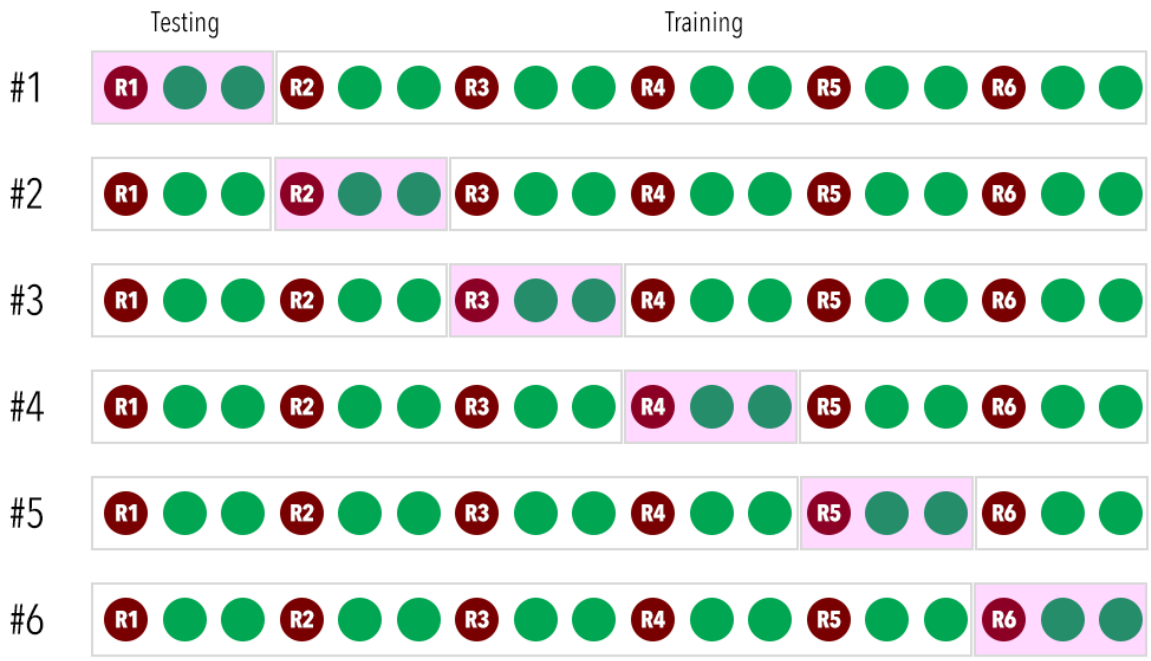

Figure 13, k-fold Cross-validation

A single log file will be formatted in CSV. An example formatted log file can be seen below:

```
1.  "Name","Count"
2.  "1002","1"
3.  "4624","2"
4.  "4627","2"
5.  "4656","94"
6.  "4658","74"
7.  "4672","2"
8.  "4673","8"
9.  "4688","12"
10. "4689","24"
11. "4690","12"
12. "4703","28"
13. "4798","4"
14. "5156","40"
15. "5158","8"
```

Type	Clean		Infected		Total	
	#	%	#	%	#	%
Original	1,000	80%	12	1%	**1,012**	**81%**
Augmented	0	0%	238	19%	**238**	**19%**
Total	**1,000**	**80%**	**250**	**20%**	1,250	100%

Table 15, Dataset Original-Augmented Distribution

Type	Clean		Infected		Total	
	#	%	#	%	#	%
Training	800	80%	200	80%	**1,000**	**80%**
Testing	200	20%	50	20%	**250**	**20%**
Total	**1,000**	**80%**	**250**	**20%**	1,250	100%

Table 16, Dataset Testing-Training Distribution

Iteration	Type	Clean		Infected		Total	
		#	%	#	%	#	%
1	Training	830	80%	208	20%	**1,038**	**83%**
	Testing	166	80%	42	20%	**208**	**17%**
2	Training	830	80%	209	20%	**1,039**	**83%**
	Testing	166	80%	41	20%	**207**	**17%**
3	Training	830	80%	208	20%	**1,038**	**83%**
	Testing	166	80%	42	20%	**208**	**17%**
4	Training	830	80%	209	20%	**1,039**	**83%**
	Testing	166	80%	41	20%	**207**	**17%**
5	Training	830	80%	208	20%	**1,038**	**83%**
	Testing	166	80%	42	20%	**208**	**17%**
6	Training	830	80%	208	20%	**1,038**	**83%**
	Testing	166	80%	42	20%	**208**	**17%**
Total		**996**	**80%**	**250**	**20%**	1,246	100%

Table 17, Dataset k-Fold CV Distribution

4.5 Feature Extraction

In order to illustrate the major difference between event occurrences of an infected log file versus that of a clean log file, the plots below were generated. The difference is calculated as follows: $abs(mean(Xi) - mean(Yi))$, where X is a vector of occurrences of an event ID i from all infected log files and Y is a vector of occurrence of an event ID i from all clean log files. Figure 15 represents the same image as Figure 14, but scaled down to the values between 0 and 60 for a better visual on the differences (see arrow for reference).

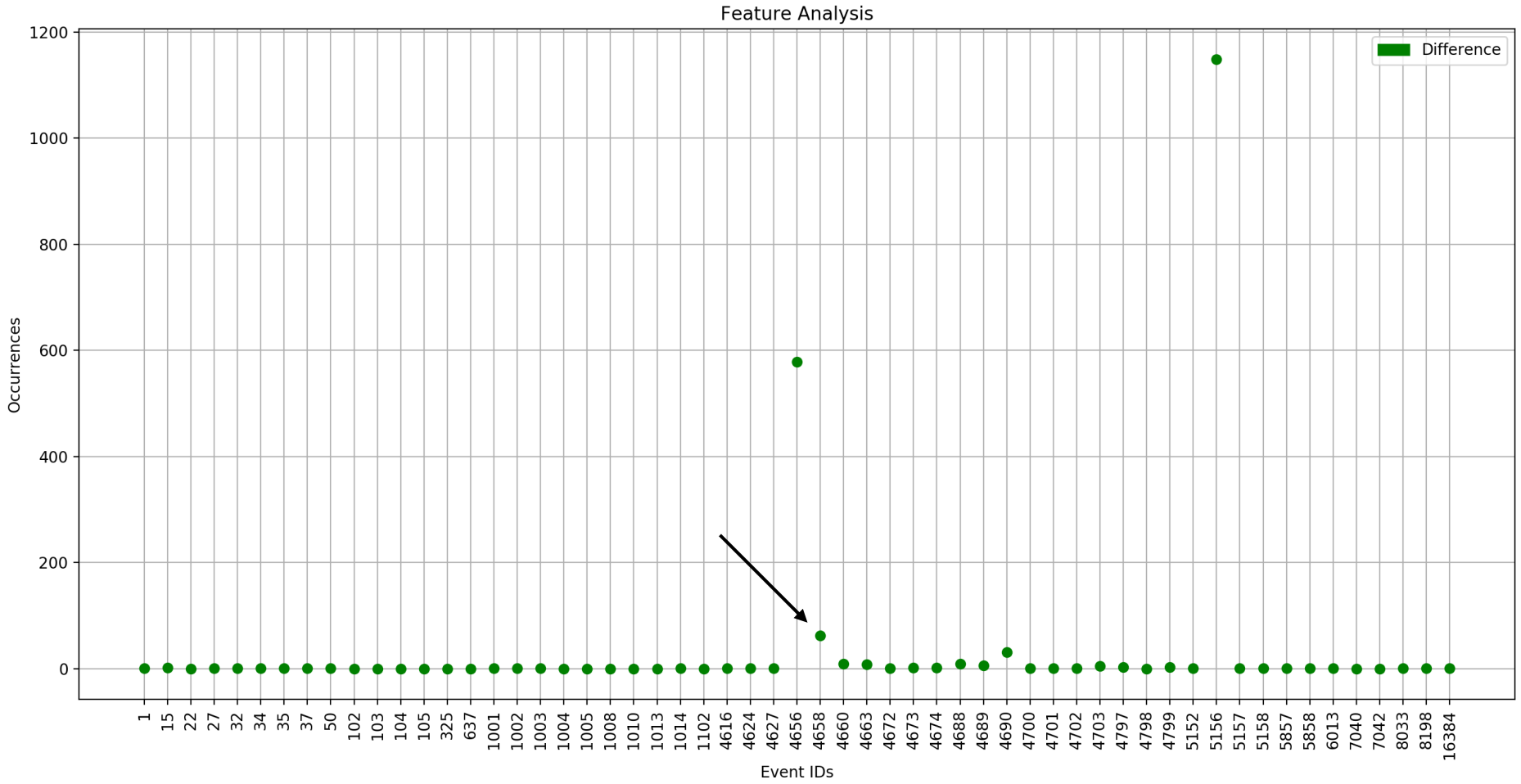

Figure 14, Feature Analysis Event IDs

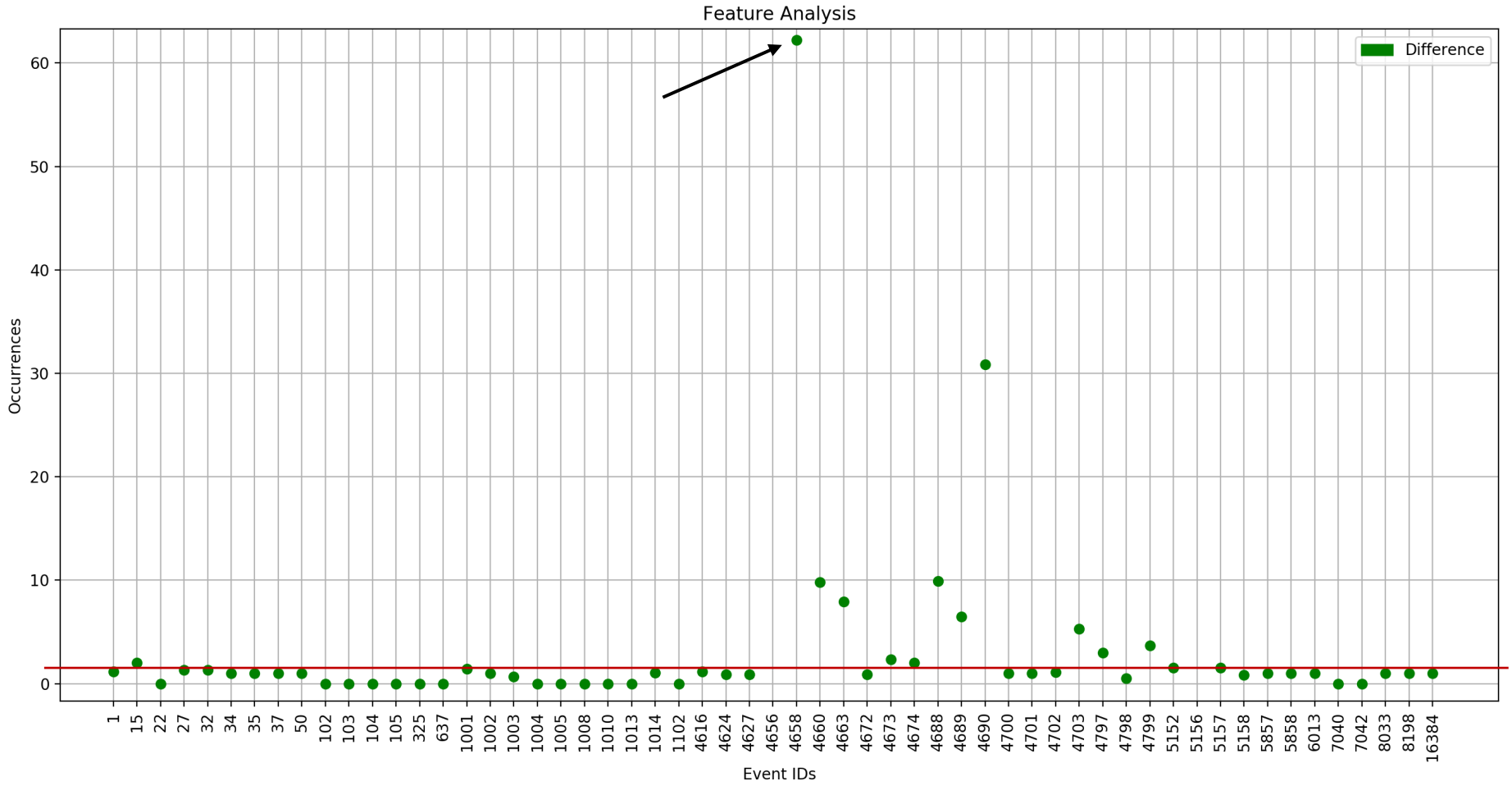

Figure 15, Feature Analysis Event IDs (zoom)

As indicated by the red line on Figure 15, there are 12 event IDs out of 57 events which correlate with typical known ransomware traits as seen in the kill chain. Although not all of the event IDs occur as frequently as others, the distinction can still be made. The descriptions in Table 18, Table 19 and Table 20 were directly acquired from (Smith, 2013).

Event ID	Description
15	Error: A disk is not ready for access yet
4656	A handle to an object was requested
4658	A handle to an object was closed
4660	An object was deleted
4663	An attempt was made to access an object
4673	A privileged service was called
4674	An operation was attempted on a privileged object
4688	A new process was created
4689	A process was exited
4690	An attempt was made to duplicate a handle to an object
4703	A user right was adjusted
4797	An attempt was made to query the existence of a blank password for an account
4799	A security-enabled local group membership was enumerated
5156	The Windows filtering platform has allowed a connection

Table 18, Feature Extraction Event IDs, source: (Smith, 2013)

Additionally, it would seem possible to specifically target the event IDs represented in Table 19. These were recommended by Randall F. Smith, president of the Monterey Technology Group Inc. and publisher of the security log encyclopaedia `www.ultimatewindowssecurity.com`.

Event ID	Description
22	Volume shadow error
4624	Successful user account login
4625	Failed user account login
4634	Remote desktop login
4740	Account lockout
4648	Account login with explicit credentials
4735	Security enabled group modification
4728	User added to privileged group
4732	
4756	

Table 19, Relevant Event IDs and Description, source: (Smith, 2013)

When using a domain controller, which will be the case in the majority of organisations, the following event IDs should be considered as well.

Event ID	Description
1102	Security log cleared
4638	Change to user (disabled, password reset, etc.)
4720	New user created
4768	Failed user account login (e.g., using Kerberos, bad password, etc.)
4771	
4776	

Table 20, Event IDs Using DC, source: (Smith, 2013)

Looking at the extracted log files from both infected and clean machines, a clear distinction can be made between the two. The occurrences for certain event IDs, often the ones referred in Table 18, are exceedingly higher in that of the infected log files. To illustrate this, 30 clean and 30 infected log files were arbitrarily picked from the dataset. The following three event IDs were specifically chosen as they were observed to occur more often within these 30 files:

- Object Access [4656]
- Privilege Usage [4673]
- Process Creation [4688]

These were then plotted for each of these log files as can be seen below. Although it shows a clear distinction between the infected and clean log files, it is also a means to understand why, when using this data format, the model could wrongly predict some of the samples. As indicated in each plot, some entries are barely indistinguishable from each other. Figure 16 depicts this very clearly; approximately a third of the of the infected log files have the same Y-value as that of the clean log files. Figure 17 displays a similar issue, four infected and four clean log files are plotted on approximately the same Y-values. This adds up to almost 15% of all the values displayed in the graph. Figure 18 is more distinguishable compared to the previous graphs, although, the first two log files have similar Y-values for that specific event. As the model will not be based on just one event occurrence, but the combination of all, the impact for some might not be as high as for others.

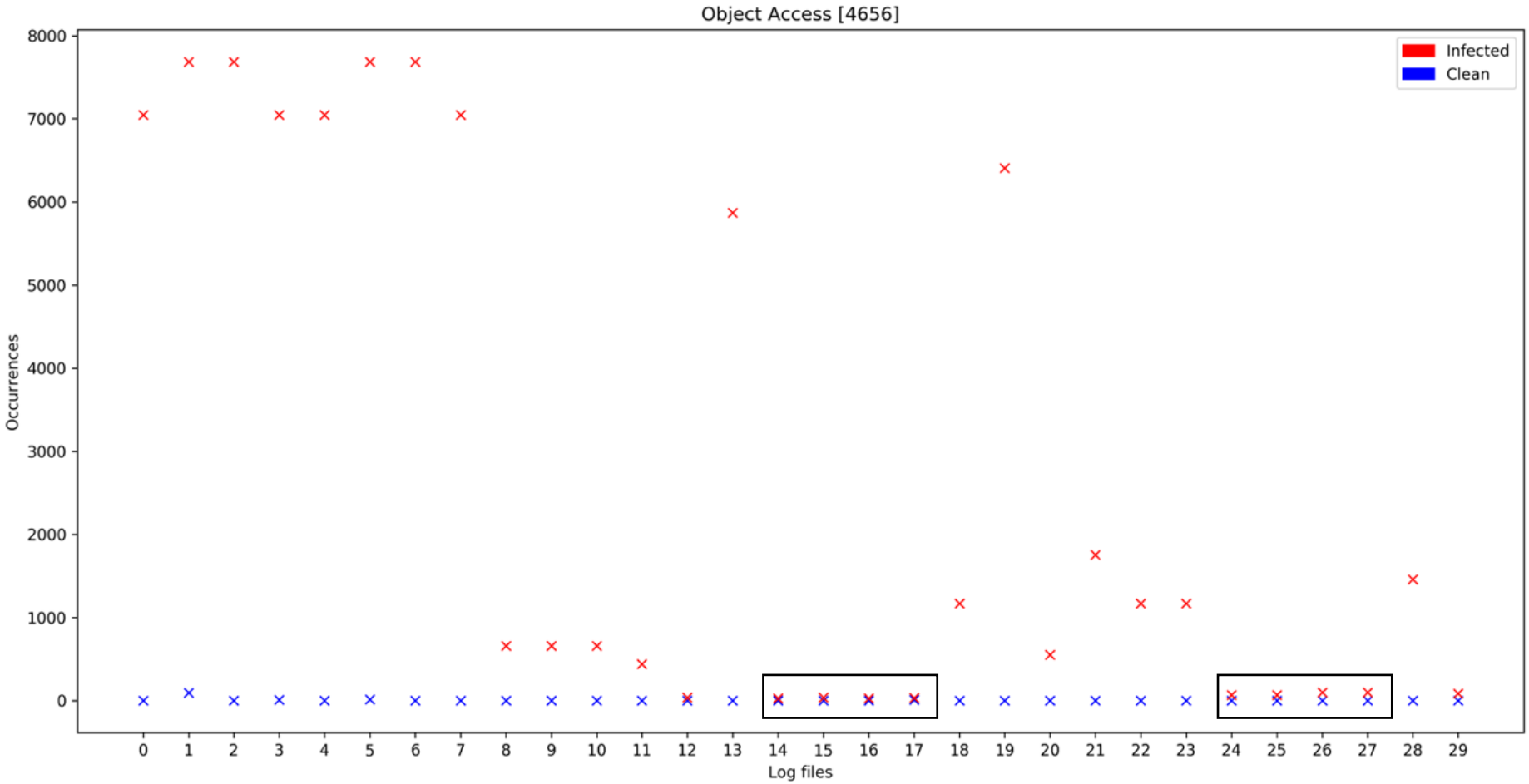

Figure 16, Object Access Plot

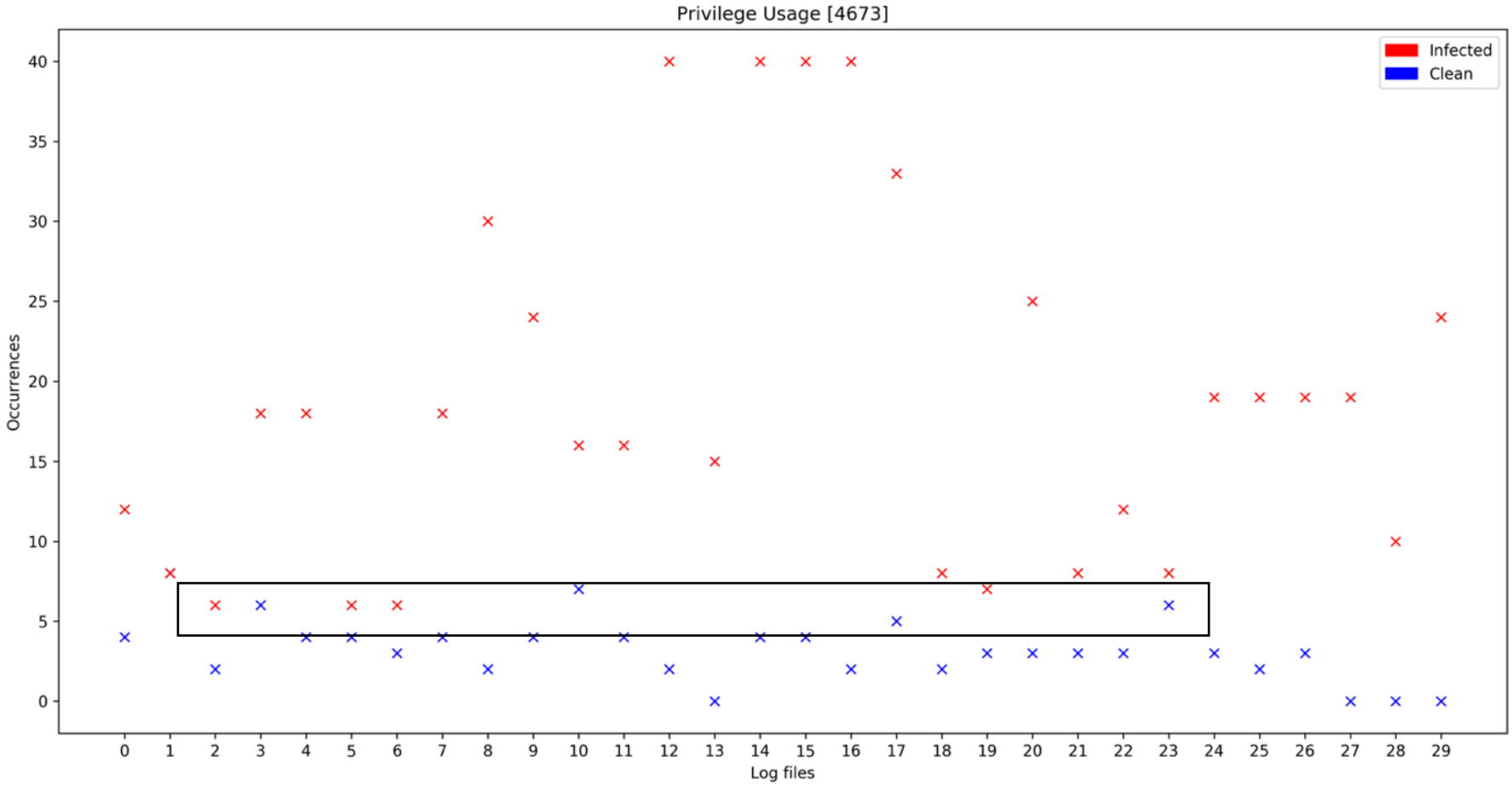

Figure 17, Privilege Use Plot

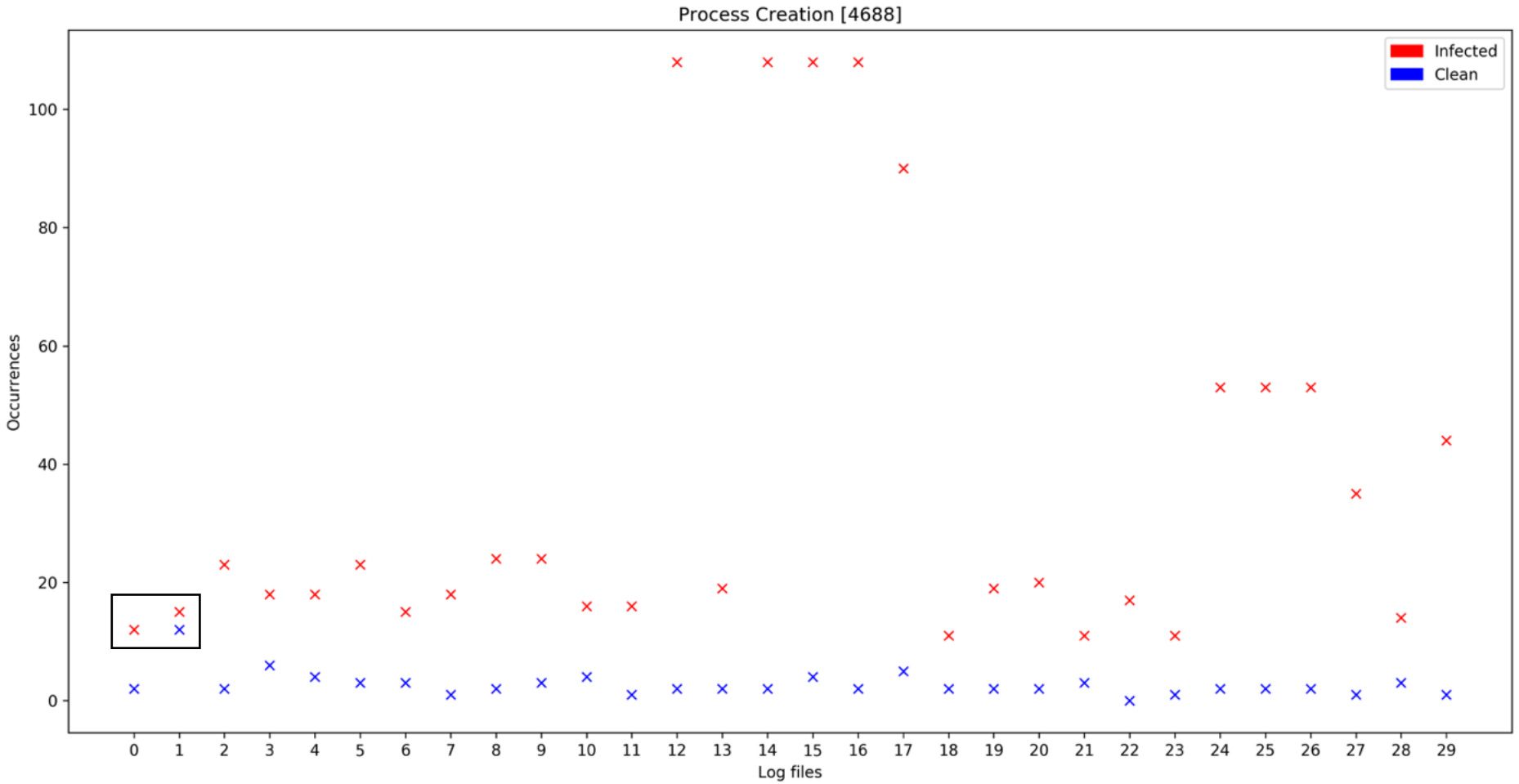

Figure 18, Process Creation Plot

4.6 Model Development

The model, as seen in Appendices G and H, consists of two classes. The KNN and AD classes contain the functionality to train and test the model using their respective algorithms as well as a feed method for real-time detection.

Both the KNN and AD classes have a common initialisation structure. First, both the training, testing and real-time dataset paths are defined in order to parse the correct log files and extract the relevant information into their corresponding variables. Next, the train and validate methods are called respectively, which then displays performance results using the output function. Finally, the feed method is called in a while loop, with a sleep timer to decrease CPU load.

Class	Method	Description
KNN	_output	Prints the FN, FP, TN, FP ratios.
	_get_occurrences	Returns a list of occurrences for each event ID.
	_get_ground_truth_list	Returns a list of target values.
	_euclidean_distance	Returns a distance between two points.
	_get_response	Returns a list of votes for a specific point.
	_get_neighbors	Returns a list of k neighbours for a specific point.
	_get_event_list	Returns a concatenated list of all event IDs used by the model. This is necessary for a consistent and fixed length input vector.
	_get_data_list	Returns a concatenated a list of all data, including file path, targets, event IDs and their occurrences. Additionally, it returns a list of just the occurrences, which is used as the input vector for the model.
	_train	Trains the model.
	_validate	Tests the model and calls the output method.
	feed	A method when looped allows for real-time detection.
AD	_output	Prints the FN, FP, TN, FP ratios.
	_get_occurrences	Returns a list of occurrences for each event ID.
	_get_ground_truth_list	Returns a list of target values extracted from the original dataset.
	_feature_normalize	Returns a normalised dataset by calculating the means and covariances of the original dataset.
	_estimate_gaussian	Returns an estimated mu (μ) and sigma (σ) vector for a given dataset.
	_multivariate_gaussian	Returns the result of the probability density function applied on a given dataset using the mu (μ) and sigma (σ) vectors.
	_get_threshold	Returns a best fit f1 score and epsilon.

	_get_event_list	Returns a concatenated list of all event IDs used by the model. This is necessary for a consistent and fixed length input vector.
	_get_data_list	Returns a concatenated a list of all data, including file path, targets, event IDs and their occurrences. Additionally, it returns a list of just the occurrences, which is used as the input vector for the model.
	_train	Trains the model.
	_validate	Tests the model and calls the output method.
	feed	A method when looped allows for real-time detection.

Table 21, Model Methods, source:(Brownlee, 2014; Shelar, 2018)

4.6.1 k-NN Algorithm

The event list is constructed by parsing the training data, testing data and real-time data log files, generating a dictionary with all the possible event IDs. This list is appended to the `self.__data_set` variable when executing the `self._get_data_list` method. The method parses all the log files and updates each event ID with their corresponding occurrence. The `self.__partial_data_set` contains a list of occurrences for each event ID. Since the `self.__data_set` list has a fixed number of events, as a result of the `self._get_event_list` method, the order does not change anymore. The `ground_truth_list` list presents the model with a list of Booleans, representing whether or not a file is infected. This is necessary to estimate the performance accuracy correctly.

```
1. self.__event_list = self._get_event_list(path_list)
2. self.__data_set, self.__partial_data_set = self._get_data_list(self.__training_path, in
   clude_target=True)
3. self.__test_data_set, self.__test_partial_data_set = self._get_data_list(self.__test_pa
   th, include_target=True)
4. ground_truth_list = self._get_ground_truth_list(self.__test_data_set)
```

The code above results into, for example:

```
event list: {7040: 0, 16384: 0, 7042: 0, 8198: 0, 4616: 0, 5857: 0, 15: 0, 4624: 0, 4627: 0, …}
data list: {'log_832.csv': {'target': 0, 'events': {7040: 7, 16384: 5, 7042: 3, 8198: 0, … }}
partial list: [[7, 5, 3, 0, …], [1, 2, 6, 0, …], … ]
testing ground truth list: [0, 0, 1, 0, 1, 1, 0, 0, 1, …]
```

Once these variables are defined, they can be passed to the k-NN algorithm. The algorithm consists of three main steps. First, the `self.__test_partial_data_set` is iterated in order to access each individual occurrence list.

```
partial list: [[7, 5, 3, 0, …], [1, 2, 6, 0, …], … ]
```

The `self._get_neighbors` method calculates the nearest `k` neighbours for point `self.__test_partial_data_set[x]`. The result of which is passed along to the `self._get_response` method, where each neighbour in the list casts a vote to determine what to classify point `self.__test_partial_data_set[x]` as. Each of these results is appended to a list of predictions.

```
k = 3
predictions = list()

for x in range(len(self.__test_partial_data_set)):
    neighbors = self._get_neighbors(self.__partial_data_set, self.__test_partial_data_set[x], k)
    result = self._get_response(neighbors)
    predictions.append(result)
```

The code above results into, for example:

```
neighbors:      [[occ_log_file_1], [occ_log_file_5], [occ_log_file_295], …]
result:         1
predictions:    [0, 0, 1, 0, 1, 1, 1, 0, 0, …]
```

The final piece of the code simply calculates the number of true positives, true negatives, false positives and false negatives by iterating over each of the predictions and comparing the result with the `ground_trust_list`.

When both the prediction and the target values are equal to 1, the true positive variable is incremented.

When both the prediction and the target values are equal to 0, the true negative variable is incremented.

When the prediction is equal to 1, and the target value is equal to 0, the false positive variable is incremented.

When the prediction is equal to 0, and the target value is equal to 1, the false negative variable is incremented.

```
1.  for index, value in enumerate(predictions):
2.
3.      if value.item() is 1 and ground_truth_list[index] is 1:
4.          tp += 1
5.      elif value.item() is 1 and ground_truth_list[index] is 0:
6.          fp += 1
7.      elif value.item() is 0 and ground_truth_list[index] is 1:
8.          fn += 1
9.      else:
10.         tn += 1
11.
12.     tp_ratio = tp / (tp + fn)
13.     fp_ratio = fp / (fp + tn)
14.     tn_ratio = tn / (tn + fp)
15.     fn_ratio = fn / (fn + tp)
16.
17. self._output(fp_ratio=fp_ratio, fn_ratio=fn_ratio, tp_ratio=tp_ratio, tn_ratio=tn_ratio
    )
```

Within the `run.py` file, the feed method is invoked within a while-loop which reads the CSV files from the `self.__real_time_path` variable. Essentially the same logic is used as in the `self._validate` method, except for the actual performance review. The additional if-statement checks whether or not it was able to read the generated CSV files correctly, as they might be encrypted during a ransomware attack. In this case, the following message is displayed: “Detected Unusual Behaviour”. Should the model succeed in parsing the CSV file(s), the entry is classified, and a prediction is made according to the k-NN algorithm. If the produced prediction vector contains the integer value 1 (classification for infection), the message “Ransomware Detected” is displayed.

```
self.__test_data_set, self.__test_partial_data_set = self._get_data_list(self.__real_time_path, real_time=True)

if self.__test_data_set:
    k = 2
    predictions = list()

    for x in range(len(self.__test_partial_data_set)):
        neighbors = self._get_neighbors(self.__partial_data_set, self.__test_partial_data_set[x], k)
        result = self._get_response(neighbors)
        predictions.append(result)

        if 1 in predictions:
            print(str(datetime.datetime.now()) + ": RANSOMWARE DETECTED. INITIATE ACTIVE DEFENCE PROCEDURES.")
        else:
            print(str(datetime.datetime.now()) + ": ENVIRONMENT IS SAFE.")
else:
    print(str(datetime.datetime.now()) + ": DETECTED UNUSUAL BEHAVIOUR.")
```

4.6.2 Outlier Detection Algorithm

The event list and both training and testing (partial) dataset variables are defined similar to that of the k-NN algorithm, as they have the same logic.

The partial dataset is passed to a `self._feature_normalize` method, which normalises the range of features as preparation for the Gaussian function. The feature set is normalised to rescale the range of, in this case, occurrences from, e.g., 0-6000 to 0-10, which is much easier to work with and plot on a graph. The mu and sigma for each occurrence list is calculated using the `self._estimated_gaussian` method, parsing the whole `self.__partial_data_set`. These are then passed to the `self._multivariate_gaussian` method, which results in a list of probabilities. The same is done to calculate the probabilities (`p_test`) of the testing dataset. Next, the `fscore` and `ep` are calculated to compare whether or not one of the values in `p` is below the threshold (`ep`). The threshold represents the barrier between a value being an anomaly or not. The result of this comparison is appended to the `outliers` list.

```
1. self.__mu, self.__sigma = self._estimate_gaussian(self._feature_normalize(self.__partia
   l_data_set))
2. self.__p = self._multivariate_gaussian(self._feature_normalize(self.__partial_data_set)
   , self.__mu,
3.                                                    self.__sigma)
4. p_test = self._multivariate_gaussian(self._feature_normalize(self.__partial_data_set),
   self.__mu, self.__sigma)
5. self.__ground_truth_set = self._get_ground_truth_list(self.__data_set)
6. self.__fscore, self.__ep = self._get_threshold(p_test, self.__ground_truth_set)
7. outliers = np.asarray(np.where(self.__p < self.__ep))
```

The code above results into, for example:

```
1. mu: [0.073, 0.085, 0.051, 0.002, …]
2. sigma: [[-0.0064, 1.0008, -0.012, …], [-0.012, 1.008, 0.9451, …], …]
3. p: [0.003, 0.0005, 0.02, 0.014, …]
4. p_test: [0.0214, 0.011, 0.0002, 0.00153, …]
5. fscore: 0.56
6. epsilon: 0.00256
7. outliers: [4, 6, 10, 18, 92, 100, …]  #indexes
```

Similar to the k-NN algorithm, the code below calculates the number of true positives and false positives by iterating over each of the `outliers` and comparing the index with the values of `self.__infected_idx_list`. When `self.__infected_idx_list` contains the `outliers` value, the true positive variable is incremented. When `self.__infected_idx_list` does not contain the `outliers` value, the false positive variable is incremented. The inverse is done for calculating false negatives. When the `outliers` list does not contain the `self.__infected_idx_list` value, the false negative variable is incremented. Lastly, the `self.__partial_data_set` is enumerated to check whether or not a value is in either of the lists. When the index is not in either the `outliers` or `self.__infected_idx_list`, the true negative variable is incremented.

```
for value in outliers[0]:
    if value in self.__infected_idx_list:
        tp += 1
    else:
        fp += 1

for value in self.__infected_idx_list:
    if value not in outliers[0]:
        fn += 1

for index, value in enumerate(self.__partial_data_set):
    if index not in outliers[0] and index not in self.__infected_idx_list:
        tn += 1

tp_ratio = tp / (tp + fn)
fp_ratio = fp / (fp + tn)
tn_ratio = tn / (tn + fp)
fn_ratio = fn / (fn + tp)

self._output(fp_ratio=fp_ratio, fn_ratio=fn_ratio, tp_ratio=tp_ratio, tn_ratio=tn_ratio
)
```

4.7 Summary

In this chapter, the *Cuckoo* environment was setup in order to deploy and analyse the six chosen ransomware samples, namely, *WannaCry*, *Cerber*, *Vipasana*, *Teslacrypt*, *Zerolocker* and *Jigsaw*. A total of 250 infected samples and 1,000 clean samples were collected and formatted

for the dataset using several *PowerShell* scripts. An in-depth analysis showed the clear distinction between infected and clean samples, justifying the feature selection. Finally, the model was developed in *Python*, implementing both the k-NN and anomaly detection algorithms.

5 Experiment Results and Analysis

This chapter provides the results of the machine learning model after evaluation. These are then further analysed to identify gaps and future research possibilities.

5.1 Results

The experiment was conducted in three phases. The first phase consisted of the setup of an isolated environment in order to safely deploy and analyse the ransomware samples, as well as generate the dataset for the model. The second phase consisted of building the model and comparing overall accuracy. Two algorithms were tested, namely k-NN and outlier detection. The model applied two variations of validation techniques on the dataset. The first one was a simple distribution of 80% training data and 20% testing data as seen in Table 16. The second was a k-fold cross-validation where the original dataset was distributed as seen in Table 17.

The results of the first evaluation were observed in a total of eight variables. Additionally, the time which was required for both algorithms to fully train and validate using the dataset was recorded as well.

Algorithm	TP	TN	FP	FN	Precision	Recall	F	OA
k-NN	98.00%	100%	0.00%	2.00%	100%	98.00%	98.99%	**99.00%**
AD	92.00%	76.62%	23.38%	8.00%	79.74%	92.00%	85.43%	**84.31%**

Table 22, k-NN and AD result comparison

TP represents the number of correct positive predictions, and TN represents the number of correct negative predictions. FP represents the number of incorrect positive predictions, and FN represents the number of incorrect negative predictions. Precision is a ratio calculated as follows: $TP / (TP + FP)$ and represents the relevance of the predictions. Recall is a ratio calculated as follows: $TP / (TP + FN)$ and represents the sensitivity or true positive rate of the predictions. The F-measure or F1-score, is a harmonic mean calculated using precision and recall: $2 * ((precision * recall) / (precision + recall))$ and represents the average of the two. Finally,

the overall accuracy is calculated as follows: $TP + TN / TP + TN + FP + FN$. Note that OA does not take into account precision, recall of F-score.

The second evaluation method performed 6 iterations, where each iteration included a training set of 5 ransomware samples and some clean data and a testing set of 1 ransomware sample and some clean data.

Algo		TP	TN	FP	FN	Precision	Recall	F	OA
k-NN	1	100%	100%	0.00%	0.00%	100%	100%	100%	**100%**
	2	0.00%	100%	0.00%	100%	80.19%	0.00%	0.00%	**50.00%**
	3	100%	99.40%	0.60%	0.00%	99.40%	100%	99.70%	**99.70%**
	4	17.07%	100%	0.00%	82.93%	100%	17.07%	29.17%	**58.54%**
	5	100%	100%	0.00%	0.00%	100%	100%	100%	**100%**
	6	0.00%	100%	0.00%	100%	79.81%	0.00%	0.00%	**50.00%**
AD	1	69.71%	84.46%	15.54%	30.29%	81.77%	69.71%	75.26%	**77.08%**
	2	100%	73.01%	26.99%	0.00%	78.75%	100%	88.11%	**86.51%**
	3	54.33%	86.14%	13.86%	45.67%	79.68%	54.33%	64.60%	**70.24%**
	4	58.85%	83.86%	16.14%	41.15%	78.47%	58.85%	67.26%	**71.35%**
	5	59.13%	84.46%	15.54%	40.87%	79.19%	59.13%	67.71%	**71.80%**
	6	58.17%	84.58%	15.42%	41.83%	79.05%	58.17%	67.02%	**71.38%**

Table 23, k-fold Cross-validation Results

	Iterations						OA
	1	2	3	4	5	6	
Algorithm	Vipasana	Zerolocker	Teslacrypt	Wannacry	Jigsaw	Cerber	Total
k-NN	100%	50%	99.70%	58.54%	100%	50%	**76.37%**
AD	77.08%	86.51%	70.24%	71.35%	71.80%	71.38%	**74.73%**
Difference	**22.92%**	**36.51%**	**29.46%**	**12.81%**	**28.20%**	**21.38%**	**1.64%**

Table 24, k-fold Cross-validation Comparison

The k-fold cross-validation for k-NN was executed within an average of 11.80 second. This includes all 41-42 samples of each ransomware. This means, the average sample was detected

within 0.28 seconds. For anomaly detection, the average was 3.27 seconds on all samples, and 0.08 seconds per sample, which is significantly lower.

Iterations							
Algorithm	Vipasana	Zerolocker	Teslacrypt	Wannacry	Jigsaw	Cerber	Inference
k-NN	11.45 sec.	11.57 sec.	12.04 sec.	12.10 sec.	11.88 sec.	11.80 sec.	**11.80 sec.**
	0.27 sec.	0.28 sec.	0.29 sec.	0.29 sec.	0.28 sec.	0.28 sec.	**0.28 sec.**
AD	3.20 sec.	3.31 sec.	3.42 sec.	3.28 sec.	3.20 sec.	3.21 sec.	**3.27 sec.**
	0.07 sec	0.08 sec.	0.08 sec	0.08 sec.	0.07 sec	0.07 sec.	**0.08 sec.**

Table 25, k-fold Cross-validation Inference Time

In the final phase, the real-time detection feature was tested. Within the same environment, the log generator script shown in Appendix D was executed in a while-loop every 15 seconds (for testing purposes). To ensure the data was accumulated accordingly, the filter and model timers were modified to 15 seconds as well. Once the log generator script and the model were running, *WannaCry* was deployed. Table 26 represents the elapsed time before the model fully detected the ransomware.

Action	Timestamp	Time Elapsed	Detection
WannaCry Started	10:20:55 AM	0 seconds	Environment is safe
Encryption Started	10:21:00 AM	5 seconds	Environment is safe
Model Detection	10:21:12 AM	17 seconds	Unusual behaviour detected
Model Detection	10:22:12 AM	82 seconds	Ransomware detected
		1 minute 22 seconds	

Table 26, Real-Time Detection Intervals Details

Figure 19 shows how the model was able to detect the ransomware within 1 minute and 22 seconds after deployment. The time stamps indicated in yellow show when the encryption process started and when the next iteration (< 15 seconds) detected unusual behaviour. The entry indicated in red show when the model fully detected the ransomware infection. Note that each deployment slightly differs based on the availability of resources, such as e.g., RAM, CPU, internet connectivity and speed. This could change the time it takes for ransomware to

fully deploy, sometimes varying between 2 to 5 minutes. Although, the detection remains approximately the same, varying only by a couple of seconds.

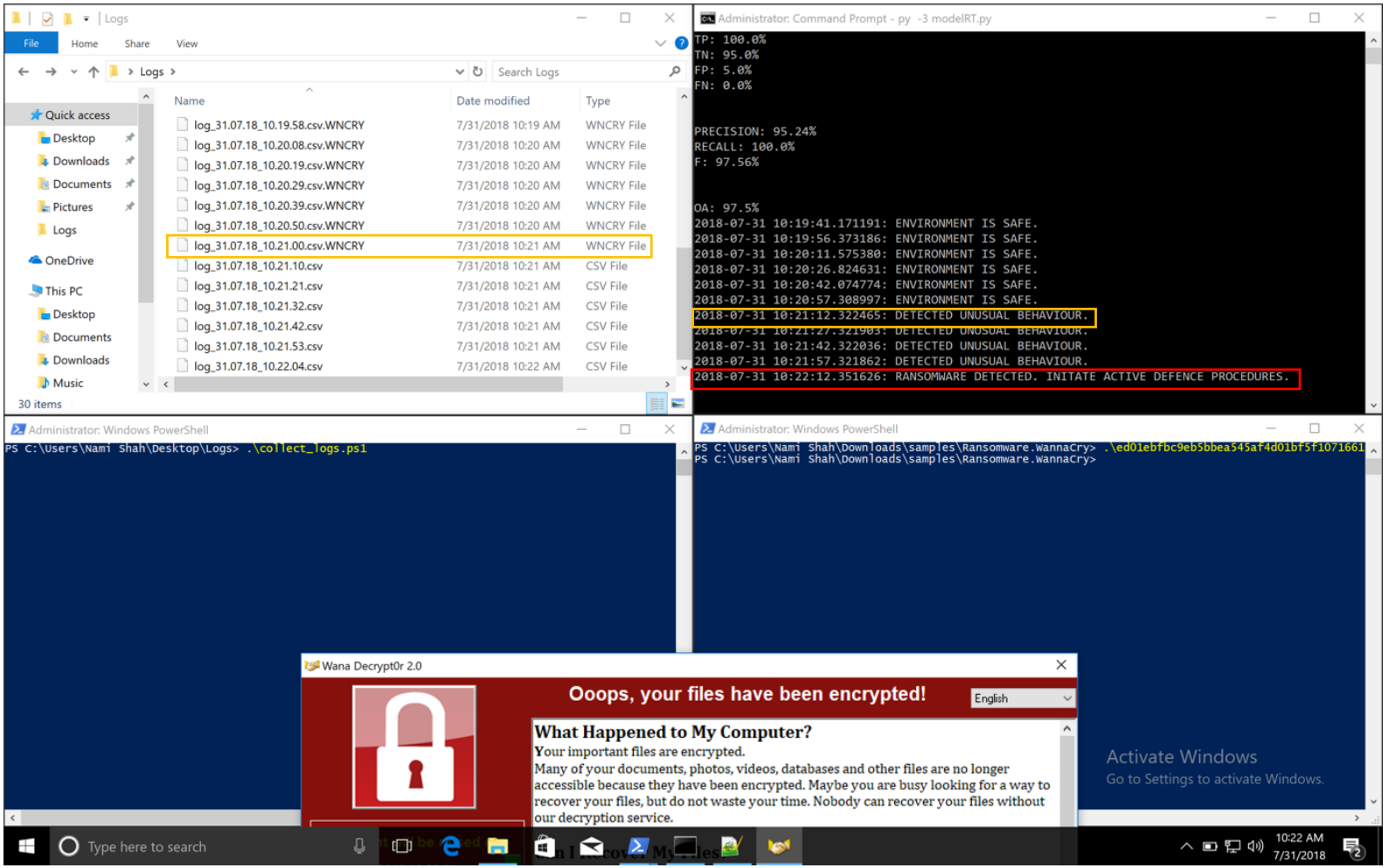

Figure 19, Demo Real-Time Detection

Finally, the model was tested with all six live samples available. Each sample was deployed three times in order to calculate an average detection time. As can be seen, the model was able to detect three of them before the encryption stage even started. Additionally, the model detected five samples within an average of 1.2 minutes, before the complete deployment of the ransomware.

Ransomware	Unusual Behaviour Detected	Ransomware Detected	Full Deployment Time
Cerber	*N/A*	46 seconds	21 seconds
Teslacrypt	*Detected before encryption*	33 seconds	2 minutes 49 seconds
Jigsaw	*Detected before encryption*	1 minute 13 seconds	3 minutes 51 seconds
WannaCry	15 seconds	1 minute 23 seconds	4 minutes 24 seconds
Vipasana	*Detected before encryption*	2 minutes 24 seconds	10 minutes 23 seconds
Zerolocker	*N/A*	1 minute 12 seconds	12+ minutes

Table 27, Real-Time Ransomware Detection Comparison

5.2 Analysis

Both the algorithms were trained and tested on the same dataset to ensure an unbiased comparison of the results. Anomaly detection was slightly faster by 1.95 seconds. K-NN took 15.66 seconds to complete training and testing, while AD took 13.71 seconds. Evidently, this number would increase or decrease depending on the size of the dataset. The k-NN detection rate for a single sample was about 0.28 seconds, whereas AD performs at 0.08 seconds for a single sample, reducing the delay of real-time detection by 3.5 times.

Overall, k-NN outperforms outlier detection in both precision and recall. The log files produced from the *Zerolocker* and *WannaCry* ransomware are relatively similar to that of a clean log file due to the low occurrences of certain events. This results in a low prediction accuracy for k-NN as the Euclidian distances (neighbours) will be similar to that of a clean log file. Although, this is where anomaly detection performed much better. Using the k-fold cross-validation technique, both algorithms achieved a similar OA. Although, outlier detection was noticeably more stable throughout the iterations, k-NN was either completely right or completely wrong in the predictions. Even though each algorithm has its own advantages and disadvantages, it would be up to the organisation to decide what type of threat they would redirect their focus

on. An unknown threat would be better handled by outlier detection, whereas existing threats would be better handled by k-NN. Taking into consideration that the dataset was not as large compared to what it could be when deployed in an organisation, the results are very positive.

Concerning real-time detection, after several deployments, the detection rate of unusual behaviour ranged between 12-20 seconds for *WannaCry*. The prediction of infection was almost always a minute later. For several samples, the ransomware was detected before the encryption stage even started. Unfortunately, the AD algorithm was not tested in real-time. The feed method for AD requires some modification in order to work, which I was unable to do due to time constraints. Real-time detection was tested using a sleep timer of 15 seconds and 30 seconds. When the timer was set to 10 seconds or lower, the model struggled to detect the ransomware within the minute. This is because the log files are extracted before any relevant activity is logged. When the timer was set to 30 seconds, the detection of unusual behaviour was prolonged accordingly, as can be expected.

The detection of unusual behaviour is based on the fact that the model is unable to read the CSV files, meaning one of two things. The user has manually modified the files to an improper read-able format, or some process has modified it. The latter is caused by the encryption stage of ransomware, indicating early signs of infection. This could be leveraged to intercept the distribution stage by blocking internet access and disconnecting the machine from any file shares and network devices. Once the infection is confirmed, a message could be sent to the SOC team, allowing them to initiate any mitigation or active defence procedures further.

5.3 Summary

Having the model evaluated provided some good insight on the strengths and weaknesses of both algorithms. Using k-fold cross-validation, k-NN was hardly able to predict the infection caused by two out of six ransomware samples. Although, when previously trained on these samples, k-NN performed near perfect, resulting in a difference of 22.63% compared to k-fold cross-validation. Anomaly detection provides less accurate results, but a more stable one with a difference of just 9.58%. Additionally, the results prove security measures can be deployed in time to stop any further damage.

6 Discussion, Conclusion and Future Work

6.1 Discussion and Limitations

During the experiment, I realised that the model might not work if the original script is encrypted during the process. Hence, two small tests were performed to verify whether or not a process keeps running even after the original script is modified. Both the tests consisted of a simple 'Hello World' output in a while-loop with a 1-second sleep timer. One was written in *Python*, the other in *PowerShell* since both will be needed for the model to work. Once the scripts were running in a privileged command prompt, two different ransomware (*WannaCry* and *Jigsaw*) samples were deployed to ensure the encryption of both the `.py` and `.ps1` files. Fortunately, both the scripts kept running in the terminal even though their original files were encrypted. This means that the code remains in memory until completely executed. Although some ransomware do not bother encrypting files smaller than a certain size, some samples do encrypt everything. This means the `filter.xml` used by the log collector script might be encrypted and thus prevent the script from running. One solution would be to add the XML content as a string within the collector script, loading it into memory altogether.

An alternative to having two scripts would be to refactor the *PowerShell* code into *Python* code and have just one script running. I believe this would be possible using the **`os.system`** command, which executes a command in a subshell. Although, this would require additional code just for formatting the output of the event viewer logs, whereas *PowerShell* has built-in functions to do exactly this. Refactoring the *PowerShell* code into *Python* would be more convenient as it would run as one background process rather than two. Additionally, a future implementation would be to adjust the feed method to work in real-time by reading newly generated files rather than being timed to read the folder every 30 seconds. This feature would also allow for a faster reaction time.

As discussed in the analysis, targeting specific event IDs, such as the ones mentioned in Table 19, might increase the accuracy of the model. My decision to focus on a more broader range of events was to make sure the model did not exclude an event which might cause a high impact

on the prediction. Since the event IDs will depend on the activity performed on the host machine. Although, reducing the scope of event IDs could be considered for future research which might focus on expanding or improving this model.

Finally, the dataset used was relatively small. Having a larger dataset available would increase the performance and accuracy of the model. Additionally, should the dataset consist of more than 50,000 files, I believe a neural network would be more appropriate as the accuracy improves based on the amount and quality of data. It would be fair to assume that organisations with several employees could easily collect this amount of data within several weeks using the provided scripts in the Appendices. An alternative to the current dataset format would be to target specific *Windows* binaries such as `lsass.exe`, or processes known to be used by ransomware based on certain keywords, similar to how AVs work using a database of known malware hashes.

A working point remains the real-time feed method of the AD class. Several code modifications are required which I was unable to do due to time constraints.

6.2 Conclusion

The model implemented as-is can identify whether or not a computer is infected by extracting a portion of the relevant log files and feeding this information into the model. The model consists of two working algorithms, namely k-NN and outlier detection, of which k-NN can detect in real-time.

I believe outlier detection would be more suitable in a professional environment as it has a better detection rate for unknown ransomware samples. Additionally, the nature of the algorithm requires a larger dataset of a certain type of activity, in this case regular activity. Which makes it more compatible for implementation in an organisation. The next step would be to automate mitigation processes such as disconnecting the infected host from the network and notify the administrator or security team of the anomaly. This can already be partially done as the model recognises the encryption stage of the ransomware, since it will not be able to

read the generated CSV files anymore. At this point, the model could force the machine to disconnect from the network and file shares. Once the encryption stage has completed, the model will be able to read the newly generated log files and detect whether or not it is infected. At this point, the model could notify the SOC team and enable active defence procedures.

One of the biggest advantages of the current model is that it provides a solid base for further research and implementation of ransomware detection using machine learning. Unfortunately, as there are no online datasets available for this model, the requirement of generating your own is quite a disadvantage, although the scripts provided in the appendix allow for somewhat automated extraction.

Finally, the combination of the dataset format and the algorithm resulted in a promising model, which I believe can be either extended or modified for other purposes rather than just ransomware detection.

Method	Area	Result	Machine Learning	Event Logs	Operating System	Real Time	Data Collection
(Furnell & Emm, 2017)	Ransomware Prevention	Methods	—	✗	—	✗	—
(National KE-CIRT/CC, 2017)	Ransomware Prevention	Methods	—	✗	—	✗	—
(Exabeam, 2016)	Ransomware Analysis	Analysis	—	✗	—	✗	—
(USMAN *et al.*, 2017)	Ransomware Analysis	Analysis	—	✗	—	✗	—
(Magklaras & Furnell, 2001)	Insider Threats	Model	—	✗	—	✓	—
(Nguyen *et al.*, 2003)	Insider Threats	Model	—	✗	—	✓	—
(Nieuwenhuizen, 2017)	Ransomware Detection	Methods	Supervised	✗	—	✗	—
(Kharraz *et al.*, 2016)	Ransomware Detection	Model	—	✗	Windows	✓	Cuckoo
(Chen & Bridges, 2017)	Ransomware Detection	Methods	—	✗	—	✗	Cuckoo
(Cabaj *et al.*, 2015)	Ransomware Analysis	Analysis	—	✗	—	✗	Maltester
(Shukla *et al.*, 2016)	Ransomware Detection	Model	—	✗	—	✓	Host
(Rieck *et al.*, 2011)	Malware Detection	Model	Many	✗	—	✓	CWSandbox
(Narudin *et al.*, 2016)	Malware Detection	Model	Anomaly Detection	✗	Android	✗	MalGenome
(Legg *et al.*, 2017)	Insider Threats	Model	Anomaly Detection	✗	—	✓	CPNI/CMU-CERT
(Agrafiotis *et al.*, 2015)	Insider Threats	Model	—	✗	—	✗	CPNI/CMU-CERT
(Stampar & Fertalj, 2015)	Intrusion Detection	Analysis	Many	✗	—	✗	—
(Soto & Zadeh, 2017)	Ransomware Detection	Model	Random Forest	✗	Windows	✓	Network
Current Method	Ransomware Detection	Model	k-NN + AD	✓	Windows	✓	Host

Table 28, Conclusion Research Gap

6.3 Future Work

The current suggested mitigation processes could be altered to allow more of an active defence approach. This was done by (Soto & Zadeh, 2017). The authors created a tool which analysed micro behaviours of network activity in order to detect malware. Moreover, they created a script which pushed GPOs to a *Windows* domain depending on specific behaviour. Essentially automating an active defence process.

I believe there are several ways to implement active defence in the current setup. For example, one way would be by mapping each event ID to a certain GPO. If the occurrence for that event ID exceeds a certain threshold, the model could enable the GPO to prevent further damage. Another way would be by having the model take into consideration the malicious process (visible in the event viewer) and blacklist the signature hash of that executable throughout the whole domain, preventing the ransomware from further executing and distributing through the network.

Furthermore, a realistic environment should be able to produce just under 1,000 normal activity log files every day per host machine (per employee). This means a log file for every 30 seconds of activity. The reason for this number is to remain alert as some ransomware can be deployed within a minute. Moreover, a sleep timer of 30 seconds is required not to stress the CPU while performing regular, daily actions on the machine. At this rate, it would be possible to use autoencoders (neural networks), which are specifically trained to output similar data as the given input data in order to learn various representations and distributions of the data. Moreover, denoising autoencoders could be used to learn the differences between corrupted versions of the data and the original data. This results in a network that can remove noise or data corruptions from the given input.

6.4 Final Remarks

Using the k-NN and anomaly detection machine learning algorithms, a ransomware detection model was developed. Comparing the accuracy and performance results of both algorithms individually, the conclusion can be drawn that k-NN provides more accurate predictions, whilst

anomaly detection provides more stable predictions. Anomaly detection outperforms k-NN when it comes to the detection of new unseen ransomware samples, whereas k-NN predicts known threats with near perfect accuracy. The real-time feature of the model could be improved by either decreasing the sleep timer when unusual behaviour is detected, or by simply making the model detect new log files the moment they are created. Additionally, an extension to the model could allow for actively implementing precautions and mitigation measures.

References

Accenture (2017) *Ransomware Incident Response and How to Avoid Future Attacks*. Available from: https://www.accenture.com/t00010101T000000Z__w__/pl-pl/_acnmedia/PDF-57/Accenture-Security-Avoiding-Ransomware.pdf (Accessed 7 June 2018).

Agrafiotis, I., Nurse, J. R. C., Buckley, O., Legg, P., Creese, S. & Goldsmith, M. (2015) Identifying attack patterns for insider threat detection. *Computer Fraud & Security*, 2015 (7): 9-17.

Ahnlab & Gartner (2015) *Ransomware Response: Ideal versus Reality*. Available from: http://www.gartner.com/imagesrv/media-products/pdf/ahnlab/ahnlab-1-2VS6RBW.pdf (Accessed 4 June 2018).

Bhatia, R. (2017) *Understanding The Difference Between Deep Learning & Machine Learning*. [online] Available from: https://analyticsindiamag.com/understanding-difference-deep-learning-machine-learning/ (Accessed 4 April 2018).

Brownlee, J. (2014) *Tutorial To Implement k-Nearest Neighbors in Python From Scratch*. [online] Available from: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/ (Accessed 23 July 2018).

Cabaj, K., Gawkowski, P., Grochowski, K. & Osojca, D. (2015) Network activity analysis of CryptoWall ransomware. *Przeglad Elektrotechniczny*, 91 (11): 201-204.

Charniak, E. & McDermott, D. (1987) *Introduction to Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc.

Chen, Q. & Bridges, R. A. (2017) Automated Behavioral Analysis of Malware A Case Study of WannaCry Ransomware. *ArXiv e-prints 1709.08753*,

Corera, G. (2017) *Kaspersky defends its role in NSA breach*. [online] Available from: http://www.bbc.com/news/technology-42009599 (Accessed 26 Mar 2018).

Crowe, J. (2017) *WannaCry Ransomware Statistics: The Numbers Behind the Outbreak*. [online] Available from: https://blog.barkly.com/wannacry-ransomware-statistics-2017 (Accessed 8 Jan 2018).

Cuckoo (2010) *Cuckoo Sandbox Book*. [online] Available from: https://cuckoo.sh/docs/ (Accessed 08 Jun 2018).

dev_jamal *I will solve machine learning problems in python*. [online] Available from: https://fiverr-res.cloudinary.com/images/t_main1,q_auto,f_auto/gigs/103555658/original/12d63b2c5f76cd827b4911901ca8b3dfc033098c/solve-machine-learning-problems-in-python.png (Accessed 4 Jul 2018).

Exabeam (2016) *The anatomy of a ransomware attack*. Available from: http://cdn2.hubspot.net/hubfs/472699/White_Paper/Exabeam_Ransomware_Threat_Report_Final.pdf (Accessed 28 Mar 2018).

Fumo, D. (2017) *Types of Machine Learning Algorithms You Should Know*. [online] Available from: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861 (Accessed 9 Aug 2018).

Furnell, S. & Emm, D. (2017) The ABC of ransomware protection. *Computer Fraud & Security*, 2017 (10): 5-11.

Grance, T., Kent, K. & Kim, B. (2004) Computer security incident handling guide. *NIST Special Publication*, 800 (61): 11.

Honeynet (2010) *GSoC 2010 Accepted Projects*. [online] Available from: https://www.honeynet.org/gsoc2010/slots (Accessed 5 Apr 2018).

IBM (2016) *Ransomware Response Guide* Available from: https://hosteddocs.ittoolbox.com/ransomware_response.pdf (Accessed 4 Jun 2018).

IC3 (2015) *Internet Crime Report*. Available from: https://pdf.ic3.gov/2015_IC3Report.pdf (Accessed 08 Jan 2018). Centre, I. C. C.

IC3 (2016) *Internet Crime Report*. Available from: https://pdf.ic3.gov/2016_IC3Report.pdf (Accessed 08 Jan 2018). Centre, I. C. C.

Kaspersky, L. (2016) *Ransomware 2016 in numbers*. [online] Available from: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2016/11/07182524/ksb_2016_eng_1.jpg (Accessed 3 Jun 2018).

Kharraz, A., Arshad, S., Mulliner, C., Robertson, W. K. & Kirda, E. (2016) UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware.

Lee, W., Stolfo, S. J., Chan, P. K., Eskin, E., Fan, W., Miller, M., Hershkop, S. & Zhang, J. (2001) Real time data mining-based intrusion detection.

Legg, P., Buckley, O., Goldsmith, M. & Creese, S. (2017) Automated Insider Threat Detection System Using User and Role-Based Profile Assessment. *Ieee Systems Journal*, 11 (2): 503-512.

Liao, Y. & Vemuri, V. R. (2002) Use of k-nearest neighbor classifier for intrusion detection1. *Computers & security*, 21 (5): 439-448.

Magklaras, G. B. & Furnell, S. M. (2001) Insider Threat Prediction Tool: Evaluating the probability of IT misuse. *Computers & Security*, 21 (1): 62-73.

Microsoft (2008a) *Troubleshooting Group Policy Using Event Logs*. [online] Available from: https://blogs.technet.microsoft.com/gpguru/2008/08/29/troubleshooting-group-policy-using-event-logs/ (Accessed 27 Apr 2018).

Microsoft (2008b) *Step-by-Step Guide to Managing Multiple Local Group Policy Objects*. [online] Available from: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc766291(v=ws.10) (Accessed 04 Apr 2018).

Microsoft (2011) *Windows Event Log*. [online] Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/aa385780(v=vs.85).aspx (Accessed 04 Apr 2018).

Nadeau, M. (2018) *11 ransomware trends for 2018*. [online] Available from: https://www.csoonline.com/article/3267544/ransomware/11-ways-ransomware-is-evolving.html (Accessed 03 Jun 2018).

Narudin, F. A., Feizollah, A., Anuar, N. B. & Gani, A. (2016) Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20 (1): 343-357.

National KE-CIRT/CC (2017) *Petya Ransomware*. Available from: http://www.ke-cirt.go.ke/files/2017/06/Petya-Ransomware-v0.3.pdf (Accessed 28 Mar 2018).

Nguyen, N., Reiher, P., Kuenning, G. & IEEE (2003) Detecting insider threats by monitoring system call activity. *Ieee Systems, Man and Cybernetics Society Information Assurance Workshop*, 45-52.

Nieuwenhuizen, D. (2017) A behavioural-based approach to ransomware detection. *MWR Labs*, 18.

Oxford (2017) *Definition of artificial intelligence*. [online] Available from: https://en.oxforddictionaries.com/definition/artificial_intelligence (Accessed 24 Dec 2017).

Pascu, L. (2017) *How Is Machine Learning Used in Bitdefender Technologies?* [online] Available from: https://businessinsights.bitdefender.com/machine-learning-bitdefender-technologies (Accessed 26 Mar 2018).

Rank Software (2017) *Cloud Security — Role of Artificial Intelligence*. [online] Available from: https://medium.com/@rank/cloud-security-role-of-artificial-intelligence-653ae0cdcf80 (Accessed 26 Mar 2018).

Rich, E. & Knight, K. (1991) Artificial intelligence.

Rieck, K., Trinius, P., Willems, C. & Holz, T. (2011) Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19 (4): 639-668.

Saunders, M. N. (2011) *Research methods for business students, 5/e*. Pearson Education India.

Savage, K., Coogan, P. & Lau, H. (2015) The evolution of ransomware.

Shelar, S. (2018) *Anomaly Detection Using Gaussian Distribution*. [online] Available from: https://www.kaggle.com/shelars1985/anomaly-detection-using-gaussian-distribution (Accessed 23 Jul 2018).

Shetty, M. & Shekokar, N. (2012) Data mining techniques for real time intrusion detection systems. *International Journal of Scientific & Engineering Research*, 3 (4): 1-7.

Shukla, M., Mondal, S. & Lodha, S. (2016) POSTER: Locally virtualized environment for mitigating ransomware threat.

Singh, N. (2016) *How to Get Started as a Developer in AI*. [online] Available from: https://software.intel.com/en-us/articles/how-to-get-started-as-a-developer-in-ai (Accessed 25 Dec 2017).

Slattery, D. (2018) *The state of cyber security in 2018: Why legacy defences won't keep pace with new ransomware and cryptojacking threats*. [online] Available from:

https://www.cso.com.au/article/641021/state-cyber-security-2018-why-legacy-defences-won-t-keep-pace-new-ransomware-cryptojacking-threats/ (Accessed 08 Aug 2018).

Smith, C., McGuire, B., Huang, T. & Yang, G. (2006) The history of artificial intelligence. *University of Washington*, 27.

Smith, F. R. (2008) *The Windows Server 2003 Security Log Revealed*. 2 edn. BookSurge Publishing.

Smith, F. R. (2013) *Security Log Quick Reference*. [online] Available from: https://cyberintruder.files.wordpress.com/2013/08/securitylogsecrets.pdf (Accessed 02 Jul 2018).

Soto, R. & Zadeh, J. (2017) Automated Prevention of Ransomware with Machine Learning and GPOs.

Stampar, M. & Fertalj, K. (2015) Artificial intelligence in network intrusion detection.

Statcounter (2018) *Desktop Windows Version Market Share Worldwide*. [online] Available from: http://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide (Accessed 06 Apr 2018).

Statista (2018) *Global market share held by operating systems for desktop PCs, from January 2013 to February 2018*. [online] Available from: https://www.statista.com/statistics/218089/global-market-share-of-windows-7/ (Accessed 27 Mar 2018).

Teplow, L. (2017) *The top 10 ransomware attacks of 2017*. [online] Available from: https://www.continuum.net/blog/the-top-10-ransomware-attacks-of-2017 (Accessed 02 Jun 2018).

Turing, A. (1950) Computing intelligence and machinery. *Mind*, 59 (2236): 433-460.

USMAN, L., PRAYUDI, Y. & RIADI, I. (2017) RANSOMWARE ANALYSIS BASED ON THE SURFACE, RUNTIME AND STATIC CODE METHOD. *Journal of Theoretical & Applied Information Technology*, 95 (11):

Von Hippel, P. (2015) *Linear vs. Logistic Probability Models: Which is Better, and When?* [online] Available from: https://statisticalhorizons.com/linear-vs-logistic (Accessed 09 Jun 2018).

Appendices

A. Host Machine Preparation

```
1. [1] $ sudo apt-get update
2. [1] $ sudo apt-get upgrade
3. [1] $ sudo apt-get install net-tools
4. [2] $ sudo apt-get install python python-pip python-dev libffi-dev libssl-dev
5. [2] $ sudo apt-get install python-virtualenv python-setuptools
6. [2] $ sudo apt-get install libjpeg-dev zlib1g-dev swig
7. [3] $ sudo apt-get install mongodb
8. [3] $ sudo apt-get install postgresql libpq-dev
9. [4] $ sudo sh -c 'echo "deb http://download.virtualbox.org/virtualbox/debian $(lsb_release -
   sc) contrib" >> /etc/apt/sources.list.d/virtualbox.list'
10. [4] $ wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | sudo apt-key add -
11. [4] $ sudo apt-get update
12. [4] $ sudo apt-get -y install gcc make linux-headers-$(uname -r) dkms
13. [4] $ sudo apt-get install virtualbox-5.1
14. [5] $ sudo apt-get install tcpdump apparmor-utils
15. [5] $ sudo aa-disable /usr/sbin/tcpdump
16. [6] $ sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
17. [6] $ getcap /usr/sbin/tcpdump
18. [7] $ git clone https://github.com/volatilityfoundation/volatility.git
19. [7] $ sudo apt-get install python-distorm3
20. [8] $ sudo apt-get install swig
```

```
[8] $ sudo pip install m2crypto
[9] $ sudo apt install libguac-client-rdp0 libguac-client-vnc0 libguac-client-ssh0 guacd
[10] $ sudo adduser cuckoo
[10] $ sudo usermod -a -G vboxusers cuckoo
[11] $ sudo pip install -U markdown
[11] $ sudo pip install -U cryptography
[11] $ sudo pip install -U pip setuptools
[11] $ sudo pip install -U cuckoo
[12] $ virtualenv venv
[12] $ . venv/bin/activate
[12] (venv)$ pip install -U pip setuptools
[12] (venv)$ pip install -U cuckoo
[13] $ sudo iptables -t nat -A POSTROUTING -o wlp3s0 -s 192.168.56.0/24 -j MASQUERADE
[13] $ sudo iptables -P FORWARD DROP
[13] $ sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
[13] $ sudo iptables -A FORWARD -s 192.168.56.0/24 -j ACCEPT
[13] $ sudo iptables -A FORWARD -s 192.168.56.0/24 -d 192.168.56.0/24 -j ACCEPT
[13] $ sudo iptables -A FORWARD -j LOG
[13] $ echo 1 | sudo tee -a /proc/sys/net/ipv4/ip_forward
[13] $ sudo sysctl -w net.ipv4.ip_forward=1
[14] $ sudo apt-get install uml-utilities bridge-utils
[15] $ sudo tunctl -b -u cuckoo -t tap_cuckoo1
[15] $ sudo ip link set tap_cuckoo1 master br0
[15] $ sudo ip link set dev tap_cuckoo1 up
[15] $ sudo ip link set dev br0 up
```

B. Custom Event Viewer Filter

```
<QueryList>
  <Query Id="0" Path="Application">
    <Select Path="Application">*[System[TimeCreated[timediff(@SystemTime) <= 30000]]]</Select>
    <Select Path="Security">*[System[TimeCreated[timediff(@SystemTime) <= 30000]]]</Select>
    <Select Path="System">*[System[TimeCreated[timediff(@SystemTime) <= 30000]]]</Select>
  </Query>
</QueryList>
```

C. Partial Cuckoo Report Output

1. Behaviour


```
{
   "behavior":{
      "generic":[
         {
            "process_path":"C:\\Windows\\System32\\lsass.exe",
            "process_name":"lsass.exe",
            "pid":600,
            "summary":{

            },
            "first_seen":1532177185.710072,
            "ppid":456
         },
         {
            "process_path":"C:\\Users\\Nami\\AppData\\Roaming\\ooojdsu.exe",
            "process_name":"ooojdsu.exe",
            "pid":3924,
            "summary":{
               "file_created":[
                  "C:\\Users\\Nami\\AppData\\LocalLow\\Microsoft\\CryptnetUrlCache\\MetaData\\E49827401028F7A0F97B5576C77A26CB_7CE95D8DCA26FE957E7BD7D76F353B08",
                  "C:\\Users\\Nami\\AppData\\LocalLow\\Microsoft\\CryptnetUrlCache\\Content\\4A13D35E1AE6430887FEE93579386ABD",
                  "C:\\Users\\Nami\\AppData\\Roaming\\log.html",
```

```
                "C:\\Users\\Nami\\AppData\\Roaming\\key.dat"
…
            ],
            "regkey_written":[
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\5.0\\Cache\\Content\\CachePrefix",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\crypto13",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\AutoDetect",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\ProxyBypass",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\5.0\\Cache\\Cookies\\CachePrefix",
                "HKEY_CURRENT_USER\\Local Settings\\MuiCache\\2\\52C64B7E\\LanguageList",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\5.0\\Cache\\History\\CachePrefix",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\IntranetName",
                "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\UNCAsIntranet"
        …
            ],
            "dll_loaded":[
                "C:\\Windows\\System32\\mswsock.dll",
                "cryptnet.dll",
                "C:\\Windows\\System32\\CRYPT32.dll",
                "C:\\Windows\\System32\\cryptnet.dll",
                "mskeyprotect.dll",
                "C:\\Windows\\SysWOW64\\cryptnet.dll"
        …
            ],
            "connects_host":[
                "7tno4hib47vlep5o.tor2web.org",
                "7tno4hib47vlep5o.tor2web.fi",
```

```
                    "7tno4hib47vlep5o.tor2web.blutmagie.de"
                ],
                "regkey_opened":[
                    "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run"
                ],
                "file_exists":[
                    "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
                    "C:\\Users\\Nami\\AppData\\Local\\Microsoft\\Windows",
                    "C:\\Users\\Nami\\AppData\\Local\\Microsoft\\Windows\\History",
                    "C:\\Users\\Nami\\AppData\\LocalLow"
            …
                ],
                "mutex":[
                    "Local\\ZonesCacheCounterMutex",
                    "System1230123",
                    "Local\\ZonesLockedCacheCounterMutex"
                ],
                "regkey_read":[
                    "HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Services\\Tcpip\\Parameters\\UseDomainNameDevolution",
                    "HKEY_LOCAL_MACHINE\\SOFTWARE\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Explorer\\FolderDescriptions\\{2B0F765D-C0E9-
4171-908E-08A611B84FF6}\\StreamResource",
                    "HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Services\\Dnscache\\Parameters\\AddrConfigControl",
                    "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Rpc\\MaxRpcSize"
            …
                ]
            }
        }
```

```
76          ]
77      }
78  }
```

2. Process Calls


```
[
  {
     "category":"network",
     "status":1,
     "stacktrace":[

     ],
     "api":"WSASocketW",
     "return_value":1156,
     "arguments":{
        "protocol":0,
        "socket":1156,
        "af":23,
        "flags":1,
        "type":2
     },
     "time":1532177191.755913,
     "tid":1432,
     "flags":{

     }
  },
  {
     "category":"file",
```

```
      "status":1,
      "stacktrace":[

      ],
      "api":"NtDeviceIoControlFile",
      "return_value":259,
      "arguments":{
         "input_buffer":"\u0003\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0019\u0000\u0000\u00c8\u0001\u0000\u0000\u00008+b\u0000\f\u0000\u0000\u0000",
         "file_handle":"0x00000484",
         "output_buffer":"\u0001\u0000\u0000\u0000x\u00f5a\u0000\u001c\u0000\u0000\u0000",
         "control_code":73919
      },
      "time":1532177191.755913,
      "tid":1432,
      "flags":{
         "control_code":""
      }
   },
   {
      "category":"synchronisation",
      "status":1,
      "stacktrace":[

      ],
      "api":"GetSystemTimeAsFileTime",
      "return_value":0,
```

```
      "arguments":{

      },
      "time":1532177191.755913,
      "tid":1432,
      "flags":{

      }
   },
   {
      "category":"system",
      "status":1,
      "stacktrace":[

      ],
      "api":"LdrLoadDll",
      "return_value":0,
      "arguments":{
         "basename":"ws2_32",
         "module_address":"0x751a0000",
         "flags":0,
         "module_name":"C:\\Windows\\system32\\ws2_32",
         "stack_pivoted":0
      },
      "time":1532177191.755913,
      "tid":1432,
      "flags":{
```

```
78
79          }
80       },
81       {
82          "category":"process",
83          "status":1,
84          "stacktrace":[
85
86          ],
87          "api":"NtCreateSection",
88          "return_value":0,
89          "arguments":{
90             "section_handle":"0x00000488",
91             "object_handle":"0x00000000",
92             "desired_access":"0x000f0005",
93             "protection":2,
94             "section_name":"",
95             "file_handle":"0x00000484"
96          },
97          "time":1532177191.771913,
98          "tid":1432,
99          "flags":{
100                "desired_access":"STANDARD_RIGHTS_REQUIRED|DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER"
101             }
102          },
103          {
104             "category":"misc",
```

```
        "status":1,
        "stacktrace":[

        ],
        "api":"SHGetFolderPathW",
        "return_value":0,
        "arguments":{
           "token_handle":"0x00000000",
           "dirpath_r":"C:\\Users\\Nami\\Desktop",
           "flags":0,
           "owner_handle":"0x00000000",
           "dirpath":"C:\\Users\\Nami\\Desktop",
           "folder":16
        },
        "time":1532177187.49254,
        "tid":8832,
        "flags":{
           "folder":"CSIDL_DESKTOPDIRECTORY"
        }
     }
  …
  ]
```

D. Generate Log PowerShell Script

```
function Generate-Log {
    $ts = Get-Date -UFormat "%d.%m.%y_%H.%M.%S"
Get-WinEvent -FilterXml ([xml](Get-Content 'filter.xml')) |
Group-Object Id |
Sort-Object Name |
Select-Object Name,Count |
Export-Csv "logs\log_$($ts).csv" -force -notypeinformation
}


    while(1) {
        Generate-Log
        start-sleep -seconds 30
    }
```

E. Random Action PowerShell Script

```
1.  $FILE_CONTENT = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac justo a nunc consequat vulputate. Sed at tincidunt enim. Curab
    itur convallis magna sed nulla luctus eleifend. Nullam ultricies nulla a sapien scelerisque, vel malesuada dui sollicitudin. Suspendisse eu nib
    h et mi gravida pharetra ac a lorem. Nunc sed mi nec magna dictum mattis. Duis pulvinar augue vitae nisl tempus molestie. Nulla eu rhoncus diam
    , vel interdum urna. In non mauris felis. Donec quis tellus nunc. Quisque facilisis tortor id quam lacinia, at sollicitudin nulla sagittis. Sus
    pendisse arcu nisl, convallis non molestie nec, imperdiet volutpat tellus. Praesent posuere massa ac risus convallis molestie…"
2.
3.  $DIRECTORY = "C:\Users\Nami Shah\Desktop\"
4.
5.  $APPLICATIONS = New-Object System.Collections.ArrayList
6.  $APPLICATIONS.AddRange( ("calc.exe","notepad.exe","mspaint.exe","explorer.exe", "cmd.exe") )
7.  $WORKING_DIR = "$($DIRECTORY)\temp"
8.
9.  function Clean-Action([String] $path = $DIRECTORY) {
10.     Try {
11.
12.         #$shell = new-object -comobject "Shell.Application"
13.         #$item = $shell.Namespace(0).ParseName("$($DIRECTORY)\Archive.zip")
14.         #$item.InvokeVerb("delete")
15.
16.
17.
18.         Remove-Item -recurse "$($path)"
19.         Remove-Item "$($DIRECTORY)\Archive.zip"
20.
```

```
21.         #Clear-RecycleBin -Force
22.
23.         New-Item -ItemType directory -Path $WORKING_DIR
24.     } Catch {
25.         Break
26.     }
27. }
28.
29.
30. function File-Action([String] $path = $DIRECTORY) {
31.     Try {
32.         $file_location = $path+"\flihgt_information.txt"
33.         $new_name = "flight_information.txt"
34.
35.         New-Item -ItemType file -Path $file_location
36.         $FILE_CONTENT > $file_location
37.         Rename-Item -Path $file_location -NewName $new_name
38.     } Catch {
39.         Break
40.     }
41. }
42.
43. function Folder-Action([String] $path = $DIRECTORY) {
44.     Try {
45.         New-Item -ItemType directory -Path $path
46.         File-Action($path)
47.     } Catch {
```

```
48.         Break
49.     }
50. }
51.
52. function Zip-Action([String] $path = $DIRECTORY, [String] $name = "Archive.zip"){
53.     Try {
54.         Folder-Action($path)
55.         Compress-Archive -LiteralPath "$($path)" -CompressionLevel Optimal -DestinationPath "$DIRECTORY\$name"
56.     } Catch {
57.         Break
58.     }
59. }
60.
61. function Delete-Action([String] $path = $DIRECTORY){
62.     Try {
63.         Folder-Action($path)
64.         Remove-Item -recurse "$($path)"
65.     } Catch {
66.         Break
67.     }
68. }
69.
70. function Recycle-Action {
71.     Try {
72.         1..10 | % {New-TemporaryFile}
73.         #explorer.exe $([io.path]::GetTempPath())
74.         dir $([io.path]::GetTempPath()) | Remove-Item -Recurse
```

```
        #(New-Object -ComObject Shell.Application).NameSpace(0x0a).Items() | Select-Object Name,Size,Path
        Clear-RecycleBin -Force
        #(New-Object -ComObject Shell.Application).NameSpace(0x0a).Items() | Select-Object Name,Size,Path
    } Catch {
        Break
    }
}

function Browser-Action([String] $url = "www.google.com"){
    Try {
        $IE=new-object -com internetexplorer.application
        $IE.navigate2($url)
        $IE.visible=$true
        Start-Sleep 5
        $IE.Quit()
        $IE = 0 # Remove .NET's reference to com object
        [GC]::collect() # run garbage collection
    } Catch {
        Break
    }
}


function Ping-Server([String] $url = "8.8.8.8"){
    Try {
           cmd.exe /c "ping $($url)"
       } Catch {
```

```
102.            Break
103.        }
104.    }
105.
106.    function App-Action([String] $appName = "explorer.exe"){
107.        Try {
108.            $proc = Start-Process "cmd.exe" -ArgumentList ("/c $($appname) && exit") -PassThru
109.            Start-Sleep 5
110.            if("$($appName)".split(".")[0] -eq "calc"){
111.                Get-Process -Name "calculator" | Stop-Process -Force
112.            } else {
113.                Get-Process -Name "$($appName)".split(".")[0] | Stop-Process -Force
114.            }
115.        } Catch {
116.            Break
117.        }
118.    }
119.
120.
121.    function Open-Apps {
122.        Try {
123.            foreach ($element in $APPLICATIONS){
124.                App-Action $element
125.            }
126.        } Catch {
127.            Break
128.        }
```

```
129.    }
130.
131.    function Main{
132.        Try {
133.            $rand = Get-Random -Maximum 10
134.
135.            $dir = "$($DIRECTORY)\stuff"
136.            switch ( $rand )
137.            {
138.                0 { File-Action $WORKING_DIR }
139.                1 { Folder-Action $dir; Clean-Action $dir }
140.                2 { Zip-Action $dir; Clean-Action $dir }
141.                3 { Recycle-Action }
142.                4 { Browser-Action }
143.                5 { Browser-Action "www.facebook.com"}
144.                6 { Ping-Server }
145.                7 { Ping-Server "127.0.0.1"}
146.                8 { Open-Apps }
147.                9 { Clean-Action $WORKING_DIR }
148.            }
149.        } Catch {
150.            Break
151.        }
152.    }
153.
154.
155.    while (1) {
```

```
156.        Main
157.        Start-Sleep 30
158.    }
```

F. Data Augmentation Script

```
1.  from pathlib import Path
2.  from tempfile import mkstemp
3.  from shutil import move
4.  from os import fdopen, remove
5.  import math
6.  import pandas as pd
7.  import random
8.
9.  SKEW = 0.2 #percent
10. DATA_PATH = "../Logs/Training Data/unskewed/"
11.
12. def normal_round(n):
13.     if n - math.floor(n) < 0.5:
14.         return math.floor(n)
15.     return math.ceil(n)
16.
17. def replace(file_path, pattern, subst):
18.     #Create temp file
19.     print(file_path, pattern, subst)
20.     fh, abs_path = mkstemp()
21.     with fdopen(fh,'w') as new_file:
22.         with open(file_path) as old_file:
23.             for line in old_file:
24.                 new_file.write(line.replace(pattern, subst))
```

```
25.     #Remove original file
26.     remove(file_path)
27.     #Move new file
28.     move(abs_path, file_path)
29.
30.
31. def parse_files(path):
32.     """Necessary parser to fill events map for all possible inputs for NN"""
33.     pathlist = Path(path).glob('*.csv')
34.     for path in pathlist:
35.         path_in_str = str(path)
36.         df = pd.read_csv(path_in_str, header=0, names=['Name', 'Count'])
37.
38.         for index, row in df.iterrows():
39.             event_id = row[0]
40.             event_occ = row[1]
41.             random_sign = bool(random.getrandbits(1))
42.             if(random_sign):
43.                 print("PLUS "+str(SKEW*100)+"%")
44.                 new_occ = normal_round(event_occ + (event_occ * SKEW))
45.             else:
46.                 print("MIN "+str(SKEW*100)+"%")
47.                 new_occ = normal_round(event_occ - (event_occ * SKEW))
48.             #print(path_in_str, event_occ, str(int(new_occ)))
49.             replace(path_in_str, ",\"" + str(event_occ) + "\"", ",\"" + str(int(new_occ)) + "\"")
50.
51.         new_path_name = path_in_str.split("_")[0] + "_skewed_" + path_in_str.split("_")[1]
```

```
52.         move(path_in_str, new_path_name)
53.
54.
55. parse_files(DATA_PATH)
```

G. Outlier Detection Model

1. ad.py

```
1.  from __future__ import division
2.
3.  from pathlib import Path
4.  from typing import List, Any, Dict
5.  import datetime
6.
7.  import numpy as np
8.  import pandas as pd
9.  from scipy.stats import multivariate_normal
10. from sklearn.metrics import f1_score
11.
12.
13. class AD:
14.     __data_set: Dict = dict()
15.     __partial_data_set = dict()
16.     __ground_truth_set: List[Any] = list()
17.
18.     __event_list: Dict = dict()
19.
20.     __p: float = None
21.     __ep: float = None
22.     __fscore: float = None
23.     __mu: float = None
```

```
24.     __sigma: float = None
25.     __infected_idx_list: List[int] = list()
26.
27.     def __init__(self, *, training_path, test_path, real_time_path):
28.         """ Initializer
29.
30.         :param training_path: absolute or relative path to training data
31.         :type training_path: str
32.         :param test_path: absolute or relative path to test data
33.         :type test_path: str
34.         :param real_time_path: absolute or relative path to real time data
35.         :type real_time_path: str
36.
37.         """
38.
39.         self.__training_path: str = training_path
40.         self.__test_path: str = test_path
41.         self.__real_time_path: str = real_time_path
42.
43.         # self._event_list = self._helper.get_event_list([self._training_path, self._cv_path])
44.
45.         # Training
46.         self._train()
47.
48.         # Validation
49.         self._validate()
50.
```

```
51.     @staticmethod
52.     def _output(*, fp_ratio, fn_ratio, tp_ratio, tn_ratio):
53.         """ Output the FP/FN/TN/TP ratios """
54.         precision = tp_ratio / (tp_ratio + fp_ratio)
55.         recall = tp_ratio / (tp_ratio + fn_ratio)
56.
57.         f = 2 * ((precision * recall) / (precision + recall))
58.
59.         total_accuracy = (tp_ratio + tn_ratio) / (tp_ratio + tn_ratio + fp_ratio + fn_ratio)
60.
61.         print("TP: " + str(np.round(tp_ratio * 100, 2)) + "%")
62.         print("TN: " + str(np.round(tn_ratio * 100, 2)) + "%")
63.         print("FP: " + str(np.round(fp_ratio * 100, 2)) + "%")
64.         print("FN: " + str(np.round(fn_ratio * 100, 2)) + "%", "\n")
65.         print("PRECISION: " + str(np.round(precision * 100, 2)) + "%")
66.         print("RECALL: " + str(np.round(recall * 100, 2)) + "%")
67.         print("F: " + str(np.round(f * 100, 2)) + "%", "\n")
68.         print("OA: " + str(np.round(total_accuracy * 100, 2)) + "%")
69.
70.     @staticmethod
71.     def _get_occurrences(data_set, *, include_target=False):
72.         """ Returns a list of occurrences for each event ID in the data set """
73.         occurrences = list()
74.         for event in data_set.values():
75.             temp = list()
76.             for occurrence in event['events'].values():
77.                 temp.append(occurrence)
```

```
78.             if include_target:
79.                 temp.append(event['target'])
80.             occurrences.append(temp)
81.         return occurrences
82.
83.     @staticmethod
84.     def _get_ground_truth_list(data_set):
85.         """ Returns a list of target values to compare with predictions """
86.         targets = list()
87.         for value in data_set.values():
88.             targets.append([value["target"]])
89.         return np.array(targets)
90.
91.     @staticmethod
92.     def _feature_normalize(data_set):
93.         """ Normalises the data set """
94.         mu = np.mean(data_set, axis=0)
95.         sigma = np.std(data_set, axis=0)
96.         return np.nan_to_num((data_set - mu) / sigma)
97.
98.     @staticmethod
99.     def _estimate_gaussian(data_set):
100.            """ Estimation of gaussian for given data set
101.
102.            Example:
103.                data_set =   -----------------> 57
104.                             |
```

```
                            |
                            |
                            |
                            |
                           1117

              data_set.T = ------------------> 1117
                            |
                            |
                            |
                            |
                            |
                            |
                            57

          """
          mu = np.mean(data_set, axis=0)  # Mean column by column
          sigma = np.cov(data_set.T)  # Cov row by row
          return mu, sigma

      @staticmethod
      def _multivariate_gaussian(data_set, mu, sigma):
          """ Returns a gaussian with a given data set, mu and sigma """
          p = multivariate_normal(mean=mu, cov=sigma, allow_singular=True)
          return p.pdf(data_set)

      @staticmethod
```

```
132.        def _get_threshold(probability, gt):
133.            """ Calculates the epsilon and f-score """
134.            best_epsilon = 0
135.            best_f1 = 0
136.            step_size = (max(probability) - min(probability)) / 1000
137.            epsilons = np.arange(min(probability), max(probability), step_size)
138.
139.            for epsilon in np.nditer(epsilons):
140.                predictions = (probability < epsilon)
141.                f = f1_score(gt, predictions, average="binary")
142.                if f > best_f1:
143.                    best_f1 = f
144.                    best_epsilon = epsilon
145.
146.            return best_f1, best_epsilon
147.
148.        def _get_event_list(self, path_list):
149.            """ Necessary parser to fill events map for all possible inputs for model """
150.
151.            try:
152.
153.                event_list = dict()
154.                for event_path in path_list:
155.                    csv_path_list = Path(event_path).glob('*.csv')
156.                    for csv_file_path in csv_path_list:
157.                        df = pd.read_csv(str(csv_file_path), header=0, names=['Name', 'Count'])
158.                        for index, row in df.iterrows():
```

```
159.                         event_id = row[0]
160.                         event_list[event_id] = 0
161.             return event_list
162.         except:
163.             #print(e)
164.             return False
165.
166.     def _get_data_list(self, path, *, include_target=False, real_time=False):
167.         """ Parsing function to iterate over training data values and push them to the model """
168.         try:
169.             path_list = [self.__training_path, self.__test_path]
170.             if real_time:
171.                 path_list.append(self.__real_time_path)
172.
173.             self.__event_list = self._get_event_list(path_list)
174.
175.             data_set = dict()
176.             csv_path_list = Path(path).glob('*.csv')
177.
178.             for csv_file_path in csv_path_list:
179.                 path_as_string = str(csv_file_path)
180.
181.                 df = pd.read_csv(path_as_string, header=0, names=['Name', 'Count'])
182.
183.                 data_set[path_as_string] = dict()
184.                 target = 1
185.
```

```
186.                 if 'log_' in path_as_string:
187.                     target = 0
188.
189.                 data_set[path_as_string]['target'] = target
190.                 data_set[path_as_string]['events'] = dict()
191.                 data_set[path_as_string]['events'].update(self.__event_list)
192.
193.                 for index, row in df.iterrows():
194.                     event_id = row[0]
195.                     event_occ = row[1]
196.                     data_set[path_as_string]["events"][event_id] = event_occ
197.
198.             return data_set, np.array(self._get_occurrences(data_set, include_target=include_target))
199.         except:
200.             #print(e)
201.             return False, False
202.
203.     def _train(self):
204.         """ Trains the model """
205.         self.__data_set, self.__partial_data_set = self._get_data_list(self.__training_path)
206.         self.__mu, self.__sigma = self._estimate_gaussian(self._feature_normalize(self.__partial_data_set))
207.         self.__p = self._multivariate_gaussian(self._feature_normalize(self.__partial_data_set), self.__mu,
208.                                                self.__sigma)
209.
210.         for index, value in enumerate(self.__data_set.values()):
211.             if value['target'] == 1:
212.                 self.__infected_idx_list.append(index)
```

```
213.
214.        def _validate(self):
215.            """ Tests the model """
216.            self.__data_set, self.__partial_data_set = self._get_data_list(self.__test_path)
217.
218.            p_test = self._multivariate_gaussian(self._feature_normalize(self.__partial_data_set), self.__mu, self.__sigma)
219.
220.            self.__ground_truth_set = self._get_ground_truth_list(self.__data_set)
221.            self.__fscore, self.__ep = self._get_threshold(p_test, self.__ground_truth_set)
222.
223.            print("F-SCORE: " + str(self.__fscore))
224.            print("Epsilon: " + str(self.__ep) + "\n")
225.
226.            outliers = np.asarray(np.where(self.__p < self.__ep))
227.
228.            self.__data_set, self.__partial_data_set = self._get_data_list(self.__training_path)
229.
230.            tp, tn, fp, fn = 0, 0, 0, 0
231.
232.            for value in outliers[0]:
233.                if value in self.__infected_idx_list:
234.                    tp += 1
235.                else:
236.                    fp += 1
237.
238.            for value in self.__infected_idx_list:
239.                if value not in outliers[0]:
```

```
240.                 fn += 1
241.
242.         for index, value in enumerate(self.__partial_data_set):
243.             if index not in outliers[0] and index not in self.__infected_idx_list:
244.                 tn += 1
245.
246.         tp_ratio = tp / (tp + fn)
247.         fp_ratio = fp / (fp + tn)
248.         tn_ratio = tn / (tn + fp)
249.         fn_ratio = fn / (fn + tp)
250.
251.         self._output(fp_ratio=fp_ratio, fn_ratio=fn_ratio, tp_ratio=tp_ratio, tn_ratio=tn_ratio)
252.         # self._helper.update_data_set('ad', [fp_ratio * 100, fn_ratio * 100, tp_ratio * 100, tn_ratio * 100])
253.
254.     def feed(self):
255.         """ Loop through this function for real-time detection """
256.
257.         self.__data_set, self.__partial_data_set = self._get_data_list(self.__real_time_path, real_time=True)
258.
259.         if self.__data_set:
260.             self.__p = self._multivariate_gaussian(self._feature_normalize(self.__partial_data_set), self.__mu,
261.                                                    self.__sigma)
262.
263.             outliers = np.asarray(np.where(self.__p < self.__ep))
264.             print(outliers[0])
265.
266.             if len(outliers[0]) > 0:
```

```
                print(str(datetime.datetime.now()) + ": RANSOMWARE DETECTED. INITIATE ACTIVE DEFENCE PROCEDURES.")
            else:
                print(str(datetime.datetime.now()) + ": ENVIRONMENT IS SAFE.")
        else:
            print(str(datetime.datetime.now()) + ": DETECTED UNUSUAL BEHAVIOUR.")
```

2. run.py

```
import warnings
import time
import matplotlib.pyplot as plt

from ad import AD

if __name__ == '__main__':

    # Ignore Warnings
    warnings.filterwarnings("ignore")

    # Suppress scientific notation
    # np.set_printoptions(suppress=True)
    # Interactive mode
    plt.ioff()

    print("Outlier Detection \n --------------------- \n\n")
    ad = AD(training_path='logs/TR/', test_path='logs/TE/', real_time_path='logs/RT/')
```

```
19.
20.     while True:
21.         ad.feed()
22.         time.sleep(30)
```

H. K-NN Model

1. knn.py

```
1.  from __future__ import division
2.
3.  import math
4.  import operator
5.  from pathlib import Path
6.  from typing import Dict
7.
8.  import datetime
9.  import numpy as np
10. import pandas as pd
11.
12.
13. class KNN:
14.     """ k-NN model """
15.
16.     __data_set: Dict = dict()
17.     __partial_data_set: Dict = dict()
18.
```

```
    __test_data_set: Dict = dict()
    __test_partial_data_set: Dict = dict()

    __event_list: Dict = dict()

    def __init__(self, *, training_path, test_path, real_time_path):
        """ Initializer """

        self.__training_path: str = training_path
        self.__test_path: str = test_path
        self.__real_time_path: str = real_time_path

        # Training
        self._train()

        # Testing
        self._validate()

    @staticmethod
    def _output(*, fp_ratio, fn_ratio, tp_ratio, tn_ratio):
        """ Output the FP/FN/TN/TP ratios """
        precision = tp_ratio / (tp_ratio + fp_ratio)
        recall = tp_ratio / (tp_ratio + fn_ratio)

        f = 2 * ((precision * recall) / (precision + recall))

        total_accuracy = (tp_ratio + tn_ratio) / (tp_ratio + tn_ratio + fp_ratio + fn_ratio)
```

```
46.
47.         print("TP: " + str(np.round(tp_ratio * 100, 2)) + "%")
48.         print("TN: " + str(np.round(tn_ratio * 100, 2)) + "%")
49.         print("FP: " + str(np.round(fp_ratio * 100, 2)) + "%")
50.         print("FN: " + str(np.round(fn_ratio * 100, 2)) + "%", "\n")
51.         print("PRECISION: " + str(np.round(precision * 100, 2)) + "%")
52.         print("RECALL: " + str(np.round(recall * 100, 2)) + "%")
53.         print("F: " + str(np.round(f * 100, 2)) + "%", "\n")
54.         print("OA: " + str(np.round(total_accuracy * 100, 2)) + "%")
55.
56.     @staticmethod
57.     def _euclidean_distance(x1, x2, length):
58.         """ Calculate distance between two points """
59.         distance = 0
60.         for i in range(length):
61.             distance += pow((x1[i] - x2[i]), 2)
62.         return math.sqrt(distance)
63.
64.     @staticmethod
65.     def _get_response(neighbors):
66.         """ Calculate how many neighbours are from specific class """
67.         class_votes = dict()
68.
69.         for x in range(len(neighbors)):
70.             response = neighbors[x][-1]
71.             if response in class_votes:
72.                 class_votes[response] += 1
```

```
73.             else:
74.                 class_votes[response] = 1
75.
76.         # Timsort O(n log n)
77.         sorted_votes = sorted(class_votes.items(), key=operator.itemgetter(1), reverse=True)
78.         return sorted_votes[0][0]
79.
80.     @staticmethod
81.     def _get_occurrences(data_set, *, include_target=False):
82.         """ Returns a list of occurrences for each event ID in the data set """
83.         occurrences = list()
84.         for event in data_set.values():
85.             temp = list()
86.             for occurrence in event['events'].values():
87.                 temp.append(occurrence)
88.             if include_target:
89.                 temp.append(event['target'])
90.             occurrences.append(temp)
91.         return occurrences
92.
93.     @staticmethod
94.     def _get_ground_truth_list(data_set):
95.         """ Returns a list of target values to compare with predictions """
96.         targets = list()
97.         for value in data_set.values():
98.             targets.append(value["target"])
99.         return targets
```

```
100.
101.        def _get_event_list(self, path_list):
102.            """ Necessary parser to fill events map for all possible inputs for model """
103.
104.            try:
105.                event_list = dict()
106.                for event_path in path_list:
107.                    csv_path_list = Path(event_path).glob('*.csv')
108.                    for csv_file_path in csv_path_list:
109.                        df = pd.read_csv(str(csv_file_path), header=0, names=['Name', 'Count'])
110.                        for index, row in df.iterrows():
111.                            event_id = row[0]
112.                            event_list[event_id] = 0
113.                return event_list
114.            except:
115.                #print(e)
116.                return False
117.
118.        def _get_data_list(self, path, *, include_target=False, real_time=False):
119.            """ Parsing function to iterate over training data values and push them to the model """
120.            try:
121.                path_list = [self.__training_path, self.__test_path]
122.                if real_time:
123.                    path_list.append(self.__real_time_path)
124.
125.                self.__event_list = self._get_event_list(path_list)
126.
```

```
127.                data_set = dict()
128.                csv_path_list = Path(path).glob('*.csv')
129.
130.                for csv_file_path in csv_path_list:
131.                    path_as_string = str(csv_file_path)
132.
133.                    df = pd.read_csv(path_as_string, header=0, names=['Name', 'Count'])
134.
135.                    data_set[path_as_string] = dict()
136.                    target = 1
137.
138.                    if 'log_' in path_as_string:
139.                        target = 0
140.
141.                    data_set[path_as_string]['target'] = target
142.                    data_set[path_as_string]['events'] = dict()
143.                    data_set[path_as_string]['events'].update(self.__event_list)
144.
145.                    for index, row in df.iterrows():
146.                        event_id = row[0]
147.                        event_occ = row[1]
148.                        data_set[path_as_string]["events"][event_id] = event_occ
149.
150.                return data_set, np.array(self._get_occurrences(data_set, include_target=include_target))
151.            except:
152.                #print(e)
153.                return False, False
```

```
154.
155.        def _get_neighbors(self, x, point, k):
156.            """ Get nearest neighbours for point from data set """
157.            distances = []
158.            for i in range(len(x) - 1):
159.                dist = self._euclidean_distance(x[i], point, len(point))
160.                distances.append((x[i], dist))
161.            distances.sort(key=operator.itemgetter(1))
162.            neighbors = []
163.            for i in range(k):
164.                neighbors.append(distances[i][0])
165.            return neighbors
166.
167.        def _train(self):
168.            """ Trains the model """
169.            self.__data_set, self.__partial_data_set = self._get_data_list(self.__training_path, include_target=True)
170.
171.        def _validate(self):
172.            """ Tests the model """
173.
174.            self.__test_data_set, self.__test_partial_data_set = self._get_data_list(self.__test_path, include_target=True)
175.
176.            k = 3
177.            tp, tn, fp, fn = 0, 0, 0, 0
178.            ground_truth_list = self._get_ground_truth_list(self.__test_data_set)
179.            predictions = list()
180.
```

```
181.            for x in range(len(self.__test_partial_data_set)):
182.                neighbors = self._get_neighbors(self.__partial_data_set, self.__test_partial_data_set[x], k)
183.                result = self._get_response(neighbors)
184.                predictions.append(result)
185.
186.            for index, value in enumerate(predictions):
187.
188.                if value.item() is 1 and ground_truth_list[index] is 1:
189.                    tp += 1
190.                elif value.item() is 1 and ground_truth_list[index] is 0:
191.                    fp += 1
192.                elif value.item() is 0 and ground_truth_list[index] is 1:
193.                    fn += 1
194.                else:
195.                    tn += 1
196.
197.            tp_ratio = tp / (tp + fn)
198.            fp_ratio = fp / (fp + tn)
199.            tn_ratio = tn / (tn + fp)
200.            fn_ratio = fn / (fn + tp)
201.
202.            self._output(fp_ratio=fp_ratio, fn_ratio=fn_ratio, tp_ratio=tp_ratio, tn_ratio=tn_ratio)
203.            # self.__helper.update_data_set('knn', [fp_ratio * 100, fn_ratio * 100, tp_ratio * 100, tn_ratio * 100])
204.
205.        def feed(self):
206.            """ Loop this function for real-time detection """
207.
```

```
208.            self.__test_data_set, self.__test_partial_data_set = self._get_data_list(self.__real_time_path, real_time=True)
209.
210.            if self.__test_data_set:
211.                k = 3
212.                predictions = list()
213.
214.                for x in range(len(self.__test_partial_data_set)):
215.                    neighbors = self._get_neighbors(self.__partial_data_set, self.__test_partial_data_set[x], k)
216.                    result = self._get_response(neighbors)
217.                    predictions.append(result)
218.
219.                if 1 in predictions:
220.                    print(str(datetime.datetime.now()) + ": RANSOMWARE DETECTED. INITIATE ACTIVE DEFENCE PROCEDURES.")
221.                else:
222.                    print(str(datetime.datetime.now()) + ": ENVIRONMENT IS SAFE.")
223.            else:
224.                print(str(datetime.datetime.now()) + ": DETECTED UNUSUAL BEHAVIOUR.")
```

2. run.py

```
1.  import warnings
2.  import time
3.  import matplotlib.pyplot as plt
4.
5.  from knn import KNN
```

```
6.
7.  if __name__ == '__main__':
8.
9.      # Ignore Warnings
10.     warnings.filterwarnings("ignore")
11.
12.     # Suppress scientific notation
13.     # np.set_printoptions(suppress=True)
14.     # Interactive mode
15.
16.     plt.ioff()
17.
18.     print("k-Nearest Neighbours \n ---------------------- \n\n")
19.
20.     knn = KNN(training_path='logs/TR/', test_path='logs/TE/', real_time_path='logs/RT/')
21.     while True:
22.         knn.feed()
23.         time.sleep(30)
```

I. Ethical Approval

wmgcourseoffice@warwick.ac.uk Inbox - Warwick 31 May 2018 at 12:39

WMG No Ethical Approval required Details

To: N.Shah.11@warwick.ac.uk, Cc: K.Debattista@warwick.ac.uk,

Reply-To: WMG Course Office Resource

Dear Mr Shah,
Warwick University ID Number: 1759576

This is to confirm that your Supervisor's Delegated Approval form has been received by the WMG FTMSc Course Office, confirming that your project: How can machine learning be applied in order to automate detection and mitigation processes of an on-going ransomware attack on a Windows environment? does NOT require ethical approval.

When you submit your project please write N/A against the ethical approval field in the submission pro-forma and include a copy of this email in the appendices of your project.

Best wishes

Jade E Barrett
WMG Full-Time MSc Course Office
wmgcourseoffice@warwick.ac.uk
go.warwick.ac.uk/wmgftmsc
+44 (0)24 7657 4206