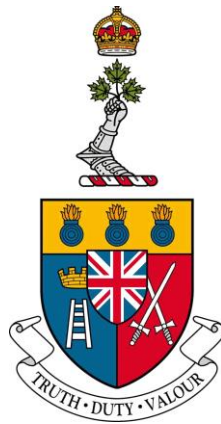


ENTERPRISE MALWARE DETECTION USING DIGITAL FORENSIC  
ARTIFACTS AND MACHINE LEARNING

DÉTECTION DE LOGICIEL MALVEILLANT DANS UN RÉSEAU  
D'ENTREPRISE À L'AIDE D'ARTEFACTS D'INVESTIGATION  
NUMÉRIQUES ET DE L'APPRENTISSAGE MACHINE



A Thesis Submitted to the Division of Graduate Studies  
of the Royal Military College of Canada  
by

Mathieu Drolet, BEng, RMC  
Major

In Partial Fulfillment of the Requirements for the Degree of  
Master of Applied Science in Computer Engineering

June, 2022

© This thesis proposal may be used within the Department of National  
Defence but copyright for open publication remains the property of the author.

## Acknowledgments

I would like to thank the Directorate Information Management Engineering and Integration (DIMEI) for the access to the test environment and the support provided by Daniel Durckovic and Gary Pecht.

In addition, I would like to thank the support provided by Naval Cadet Cody Carter who supported me with the environment setup and the user simulation, and Lieutenant Ryan Tsui for the custom malware samples he provided.

## Abstract

Network defense is a complex field. Intrusions can come from multiple vectors and leverage any number of vulnerabilities in a computer system. Over the years, technologies have been developed to reduce outside threats. Intrusion detection systems and log aggregation solutions have been introduced to help analysts find anomalies and detect intrusions. These systems can make use of agents deployed across a network to monitor the state of the computers. On the hosts, multiple techniques can be used to detect intrusions, such as signature or anomaly-based detection algorithms. However, those solutions need a lot of tuning, skilled analysts and modern equipment. Processing host-based data is challenging, as every log, event and configuration can be looked at; this generates a significant amount of data.

In digital forensics, the objective is to look at a system's state such as its processes in memory, as well as artifacts left on the disk. This can then be used to build a timeline of events to detect and understand how a malware which compromised a workstation works, so countermeasures can be implemented. By automating this process across an entire network of live computers, it would be possible to analyze the state of those systems to look for anomalies. However, this creates significant amount of data that needs to be collected, transferred and processed. It can also require significant resources. Previous research using memory forensics have looked at host-based artifacts from the memory capture collected using virtual machine snapshots to detect malware such as rootkit and ransomware. The main challenge for remote forensic analysis is timely acquisition of the memory and its analysis.

This thesis aims to propose an automated solution based on digital forensics and machine learning to detect intrusions proactively across multiple computers. The

Digital Forensic – Incident Response (DFIR) tool Velociraptor was identified as the perfect tool to collect the artifacts that would be analyzed by a machine learning model. It is a lightweight platform that collects disk-based information and volatile information using the Windows API. The artifacts were collected and then analyzed by three machine learning algorithms, Isolation Forest, Random Forest and Support Vector Machine, to determine if anomalous activities were ongoing on the computer. Using this technique, Random Forest and Support Vector Machine achieved a perfect classification of the 41 malware samples used for the experiment.

## Résumé

La défense des réseaux est un domaine complexe. Les intrusions peuvent provenir de plusieurs vecteurs et exploiter un certain nombre de vulnérabilités dans un système informatique. Au fil des ans, des technologies ont été développées pour réduire les menaces externes. Des systèmes de détection d'intrusion et des solutions d'agrégation de journaux d'événement ont été introduits pour aider les analystes à trouver des anomalies et à détecter des intrusions. Ces systèmes peuvent utiliser des agents déployés sur un réseau pour surveiller l'état des ordinateurs. Sur les postes de travail, plusieurs techniques peuvent être utilisées pour détecter les intrusions, telles que des algorithmes de détection basés sur des signatures ou des anomalies. Cependant, ces solutions nécessitent beaucoup de réglages, des analystes qualifiés et des équipements modernes. Le traitement des données disponible sur un ordinateur est complexe, car chaque journal, événement et configuration peut être examiné; cela génère une quantité importante de données.

En investigation numérique, l'objectif est d'examiner l'état d'un système tel que ses processus en mémoire, ainsi que les artefacts laissés sur le disque. Cela peut ensuite être utilisé pour créer une chronologie des événements afin de détecter et de comprendre comment les logiciels malveillants ont compromis un poste de travail, afin que des contre-mesures puissent être mises en œuvre. En automatisant ce processus sur l'ensemble d'un réseau d'ordinateurs actifs, il serait possible d'analyser l'état de ces systèmes pour rechercher des anomalies. Cependant, cela crée une quantité importante de données qui doivent être collectées, transférées et traitées. Cela peut aussi nécessiter des ressources importantes. Des recherches antérieures utilisant l'analyse de la mémoire d'un ordinateur ont examiné les artefacts basés sur l'hôte à partir de la capture de mémoire collectée depuis la mémoire d'une machine virtuelle pour détecter les logiciels malveillants tels que les programmes

malveillants furtifs et les logiciels de rançon. Le principal défi d'investigation numérique à distance est l'acquisition rapide de la mémoire et son analyse.

Cette thèse vise à proposer une solution automatisée basée sur l'investigation numérique et l'apprentissage machine pour détecter les intrusions de manière proactive sur plusieurs ordinateurs. Velociraptor a été identifié comme l'outil parfait pour collecter les artefacts qui seraient analysés par un modèle d'apprentissage machine. Il s'agit d'une plate-forme légère qui collecte des informations disponibles sur le disque et dans la mémoire à l'aide de l'interface de protocole d'application Windows. Les artefacts ont été collectés puis analysés par trois algorithmes d'apprentissage machine, forêts isolées, forêts aléatoires et machine à vecteurs de support, afin de déterminer si des activités anormales étaient en cours sur l'ordinateur. Grâce à cette technique, les algorithmes forêts aléatoires et machine à vecteurs de support ont réalisé une classification parfaite des 41 échantillons de logiciels malveillants utilisés pour l'expérience.

# Table of Contents

Acknowledgments.....	ii
Abstract .....	iii
Résumé.....	v
Table of Contents.....	vii
List of Tables .....	xi
List of Figures.....	xii
Section 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Statement of Deficiency .....	2
1.3 Aim .....	3
1.4 Research Activities .....	4
1.5 Organization .....	5
Section 2 Literature Review .....	6
2.1 Performance .....	6
2.2 Machine Learning.....	8
2.3 Intrusion Detection System (IDS).....	10
2.3.1 Cyber Kill Chain.....	11
2.3.2 Network Intrusion Detection Systems (NIDS) .....	13
2.3.3 Host-Based Intrusion Detection Systems (HIDS).....	15
2.3.4 Data Analysis Techniques .....	16

2.4	Enterprise Network Monitoring Solutions .....	22
2.5	Memory Forensics .....	24
2.6	Limitations .....	27
Section 3 Background .....		29
3.1	Machine Learning .....	29
3.1.1	Algorithms.....	29
3.1.2	Feature Reduction .....	35
3.1.3	Feature Selection.....	37
3.1.4	Scaling.....	38
3.1.5	Parameter tuning.....	39
3.2	Malware .....	46
3.2.1	Remote Access Trojan .....	46
3.2.2	Rootkit.....	47
3.2.3	Ransomware .....	50
3.2.4	Fileless Malware .....	50
3.3	Persistence Mechanisms.....	52
3.4	Velociraptor.....	55
Section 4 Methodology .....		57
4.1	Velociraptor and Volatility Methods .....	57
4.2	Data Acquisition.....	58
4.2.1	Test Environment.....	58
4.2.2	Malware.....	63
4.2.3	Features Extraction .....	73



4.2.4	Data Collection .....	79
4.2.5	Experimentation.....	81
4.3	Exploratory Data Analysis and Data Munging.....	83
4.3.1	Data Munging .....	83
4.3.2	Volatility .....	84
4.3.3	Velociraptor .....	88
4.4	Feature Engineering .....	92
4.4.1	Feature Reduction .....	92
4.4.2	Cross-Validation .....	92
4.5	Model Learning and Evaluation .....	93
4.5.1	Model Selection .....	93
4.5.2	Hyperparameter tuning.....	94
4.5.3	Model Training .....	96
Section 5 Results.....		99
5.1	Results.....	99
5.1.1	Performance Metrics .....	99
5.1.2	Isolation Forest .....	100
5.1.3	Random Forest.....	105
5.1.4	Support Vector Machine .....	107
5.2	Missed Malware .....	109
5.3	Discussion .....	111
5.3.1	Volatility Performance .....	111
5.3.2	Perfect Classification .....	114

5.4	Validation.....	115
Section 6 Conclusion.....		117
6.1	Contribution .....	118
6.2	Future Work .....	118
References .....		120
Annex A Velociraptor Features List .....		A-1

## List of Tables

Table 1—User workstations in the test environment .....	60
Table 2—Volatility workstations schedules .....	61
Table 3—Velociraptor workstations schedules .....	61
Table 4—Malware executables and tools used.....	63
Table 5—Features from Cohen and Nissim [4] (Reproduced from [4]) .....	74
Table 6—Features from Wang et al. [5] used in this thesis (Reproduced from [5]) .....	74
Table 7—Features added based on Parker and Nissim [6].....	75
Table 8—Features considered by Mohammad and Alqahtani [29] (Reproduced from [12]) .....	77
Table 9—Features considered by Murthaja et al. [2] (Reproduced from [2]) .....	78
Table 10—Type of features generated for Velociraptor .....	79
Table 11—Velociraptor features selected using PPS.....	89
Table 12—Trained model combinations .....	97
Table 13—Results using Isolation Forest with Volatility .....	102
Table 14—Results using Isolation Forest with Velociraptor.....	104
Table 15—Results using Random Forest with Volatility .....	106
Table 16—Results using Random Forest with Velociraptor .....	107
Table 17—Results using SVM with Volatility .....	108
Table 18—Results using SVM with Velociraptor .....	109
Table 19—Volatility malware type detection, by algorithms.....	110
Table 20—Velociraptor malware type detection, by algorithms .....	111
Table 21—Comparison of the results obtained with Volatility with the works from Cohen and Nissim [4] and Wang et al. [5] .....	112
Table 22—Comparison of Volatility and Velociraptor, for each algorithm.....	116
Table A-1—Complete Velociraptor features list .....	A-1

## List of Figures

Figure 1—ROC curve .....	8
Figure 2—Anomaly detection (Reproduced from [7]).....	9
Figure 3—Machine Learning Pipeline (Reproduced from [9]) .....	10
Figure 4—Classifications of intrusion detection systems (Reproduced from [14])	11
Figure 5—Lockheed Martin kill chain (Reproduced from [13]) .....	13
Figure 6—Data analysis techniques (Derived from [14]) .....	17
Figure 7—Detection results of the various EDRs tested by Karantzas and Patsakis [35] using CPL, HTA, EXE and DLL attacks (Reproduced from [35]) ...	24
Figure 8—Some features generated by Cohen and Nissim [4] research (Reproduced from [4]) .....	26
Figure 9—Volatility psxview plugin output (Reproduced from [38]) .....	27
Figure 10—Isolating data points using Isolation Forest. $X_i$ (left) is more normal than $X_0$ (right) (Reproduced from Liu et al. [42]) .....	30
Figure 11—Random Forest ensemble tree prediction (Reproduced From Yiu [45]) .....	32
Figure 12—SVM attempting to find the best hyperplane (Reproduced from [48])	33
Figure 13—Soft Margin SVM (Reproduced from [48]) .....	34
Figure 14—Linear and Non-linear SVM (Reproduced from [48]).....	35
Figure 15—Subspace selection using PCA (Reproduced from [7]) .....	36
Figure 16—PCA reduction to retain 95% of the variation, using the SKLearn library .....	36
Figure 17—Correlation matrix vs. PPS matrix (Reproduced from [54]) .....	38
Figure 18—SVM sensitivity to scaling (Reproduced from [7]) .....	39
Figure 19—Example of overfitted data (Reproduced from [7]) .....	40
Figure 20—Bagging used to train a machine learning model (Reproduced from [7]) .....	41

Figure 21—AdaBoost, a common boosting algorithm (Reproduced from [7]).....	42
Figure 22—Comparison of Grid Search and Random Search (Reproduced from [61]) .....	43
Figure 23—Example of a 5-fold Cross-Validation (Reproduced from [67]) .....	45
Figure 24—Cross-Validation flowchart (Reproduced from [67]) .....	45
Figure 25—RAT Structure .....	47
Figure 26—Kernel-level rootkit implementation (Reproduced from [74]).....	49
Figure 27—Infection flow of files malware (Reproduced from [77]) .....	51
Figure 28—Various persistence mechanisms used by malware (Reproduced from [79]) .....	52
Figure 29—User-level persistence registry keys .....	53
Figure 30—Administrator-level persistence registry keys .....	53
Figure 31—AppInit_DLL Registry key (Reproduced from [81]) .....	54
Figure 32—Actions if a service fails to start (Reproduced from [84]) .....	55
Figure 33—Test Environment .....	59
Figure 34—CatfishHTTPSExfiltrator server receiving files on the server .....	64
Figure 35—Files on the victim computer after Lionfish encrypted the files in the user’s Document folder .....	65
Figure 36—CatfishFileShredder trojan window.....	65
Figure 37—Process migration using Meterpreter .....	67
Figure 38—Cobalt Strike main interface (Reproduced from [93]).....	69
Figure 39—OffensivePH killing a process.....	71
Figure 40—PEview of Detours with a rogue DLL, evil.dll (Reproduced from [23]) .....	72
Figure 41—Hidden rootkit hiding a file.....	73
Figure 42—Experiment Flow .....	82
Figure 43—Volatility features importance, using SKLearn <i>feature_importances</i> class.....	84
Figure 44—Volatility features Correlation matrix.....	86

Figure 45—Volatility features PPS matrix.....	87
Figure 46—Velociraptor features importance, using SKLearn <i>feature_importances</i> class.....	88
Figure 47—Velociraptor features correlation matrix.....	90
Figure 48—Velociraptor features PPS matrix.....	91
Figure 49—Comparison of the effectiveness of multiple machine learning algorithms for a file system analysis (Reproduced from Mohammad and Alqahtani [12])......	94
Figure 50—Isolation Forest Grid Search.....	95
Figure 51—Random Forest Grid Search.....	95
Figure 52—SVM Grid Search.....	96
Figure 53—Summary of the methodology used for both methods to train the machine learning models.....	98
Figure 54—PCA reduction to two dimensions of the Volatility entire dataset, with normal data points shown in blue and malware points shown in orange.....	103
Figure 55—PCA reduction to two dimensions of the Velociraptor entire dataset, with normal data points shown in blue and malware points shown in orange.....	105

## Section 1

# Introduction

Malware detection is a difficult task. Malware can come in many shapes, target different types of processes, protocols and devices. In order to defend against intrusions, three different strategies are typically used. The first one is trying to detect the malware as it is entering the network or the computer. This is why a lot of research has been focused on the detection of file downloads from suspicious URL or a user browsing to a malicious website. However, URLs can easily be changed to look less suspicious or avoid domains that have been blacklisted. This method can also be applied to detect internal communications once the adversary has entered the network. The second one is looking at what is happening inside the workstations, such as the series of system calls made in the operating system (OS), and what processes have been launched and by which parent process. This can also include anti-virus software, which compares specific strings and binary patterns to a database of known malicious software. However, some malware, such as rootkits, affect the functioning of the OS and hide from anti-virus software, making detection more difficult. Lastly, digital forensic is a field that looks at the compromised computer hard-drive and memory, post-exploitation, to better understand how the malware works, so signatures can be developed to detect a subsequent occurrence of the intrusion. The advantage of digital forensic is that it can provide the most context on the state of the machine at the moment when the disk or the memory capture was collected. Files that have been deleted or hidden can often be retrieved and lead to malware analysis and a deeper understanding of the adversary and its intent. This process, however, takes time and requires a skilled analyst.

### 1.1 Motivation

When a digital forensic investigation begins, the evidence of malicious activity on the compromised computer may have been hidden or deleted by the adversary in an attempt

to hide its presence. The sooner the forensic collection happens, the better the chances of collecting relevant artifacts. In addition, volatile memory is difficult to recover in a timely manner, as a memory capture must be done and extracted while the malware is running. Being able to collect, at a regular interval, digital evidence remotely, including from the volatile memory, would create more accurate and relevant collection of digital evidence. However, this brings the challenge of having too much data. One way to make sense of all this data is to use a machine learning algorithm. By training a model with clean computers, a baseline can be established. From there, all new collections can be analyzed and classified. Performing an automated digital forensic investigation remotely across multiple hosts creates a new way of conducting host-based monitoring, using a different approach and different artifacts.

The main motivation behind this thesis is therefore to propose a method of detecting malware based on digital forensic techniques, host-based monitoring systems and machine learning techniques.

## 1.2 Statement of Deficiency

Current host-based monitoring solutions, such as Endpoint Detection and Response (EDR) tools and anti-virus software, use agents, which are located on the endpoints and report back to a central server. Depending on how well this data is collated with network-based tools, it can be difficult to understand the broader picture. In addition, they can generate a large number of alerts as they have the ability to report on many events and logs. Furthermore, EDRs do not necessarily have the resources to perform the behavioural analysis required to detect new threats as it requires more central processing unit (CPU) resources; they are more suited to detect known threats, which compose the majority of the threats faced by enterprise networks (86%) [1]. Security Operation Centers (SOC) need to rely on additional tools, such as threat hunting feeds and indicators of compromises (IOC).

In order to get accurate behaviour-based detection, a good baseline, which has enough samples, and captures the normal network and computer activity, is required. The



challenge is, however, that the network requirements and software used by the users evolve over time. The baseline needs to be able to adapt to maintain efficiency.

The focus of current digital forensic methods is finding and understanding how malware got into a system so that some mitigating actions can be implemented to prevent further intrusions. Although this provides good intelligence, it is usually not timely. Current research using digital forensics focus on either disk-based forensics or memory forensics. This thesis intends to improve from those methods by using both types of artifacts to have a broader view over a host. Using the Windows API, volatile information generally found in memory can be obtained rapidly to supplement data found on the disk. Current memory forensics methods use VM snapshots or sandbox environment to acquire the memory. This approach works well in a test environment, but cannot be applied to an enterprise network that uses physical workstations. A different acquisition method is required.

Once artifacts are collected, making sense of the data is a significant challenge due to the sheer size of the collection. Applying machine learning to automate a digital forensic investigation is a novel idea. Most research focuses on extracting forensic artifacts from a single computer or a sandbox, but not from a live enterprise system [2], [3]. In order to apply the principles and findings from [2], [3] in a live network, the main change required is the collection and analysis of artifacts in near real-time. This thesis aims to create a solution that can be applied to live enterprise networks.

### 1.3 Aim

The aim of this thesis is to propose an automated solution based on digital forensics and machine learning to detect intrusions proactively across multiple computers.

My research hypothesis is that using both types of artifacts, disk-based and memory-based artifacts, provides a better view over the activities currently ongoing on the computer.

The implemented method is validated against the memory forensics research from Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6], using machine learning

algorithms and memory forensic techniques; while Velociraptor is a DFIR tool with access to both disk-based and volatile artifact and Volatility is a memory forensics only tool, this is the most similar research approach. From their research, 40 features generated from Volatility plugins were selected to train the machine learning models. The validation objective is to get a better classification with the implemented methodology this validation methodology based on [4], [5] [6]. In this thesis, those two methodologies are referred to as Velociraptor and Volatility, respectively.

#### 1.4 Research Activities

The following activities were conducted in order to achieve the research aim:

- Set up the test network;
- Determine artifacts and features to be collected;
- Simulation of user activity on the network;
- Collect forensic artifacts on clean computers to build a baseline of normal behaviour;
- Run malware and collect artifacts from the compromised system;
- Implement three different machine learning algorithms separately to model normal and detect anomalies;
- Compare three machine learning algorithms to determine which one is better for the scope of this research; and
- Compare the work done by Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6], which used Volatility and machine learning, to my method using Velociraptor and machine learning. The implemented method uses both types of artifacts, disk-based and volatile artifacts, to improve from [4], [5] and [6].

## 1.5 Organization

This document is divided in six sections. First, Introduction, which has been covered. Second, Literature Review, which covers recent research relevant to this thesis with their limitations. Third, Background, which covers the theory and technical concepts relevant to this research. Forth, Methodology, which explains the research activity steps and design decision used during this research. Fifth, Results, which presents and discuss the results of this thesis. Finally, the Conclusion, which covers potential future work.

## Section 2

# Literature Review

Ever since computer networks started to be targeted, network administrators and security practitioners have been working at securing networks and detecting intrusions as quickly as possible. Any delay in response gives an intruder precious time to perform actions on objectives, such as collecting and exfiltrating data, modifying files or deleting information.

This led to the emergence of multiple fields: network traffic analysis to attempt to detect anomalous inbound and outbound traffic, host-based monitoring to detect anomalous activity on a running computer, and digital forensic to investigate past compromises to learn from them and help closing gaps.

Development of anti-virus software and host-based monitoring solutions look at the activities happening on a computer, to either compare a running program to a database of known malicious software using signatures, or attempt to see patterns that deviate from a specific baseline to find behavioural anomalies. This section discusses past research that looked into this issue and will provide some background for this thesis.

### 2.1 Performance

In order to understand how research results are compared in the next section, the performance metrics used in machine learning need to be understood.

Performance of a machine learning algorithm can be evaluated using multiple metrics, such as accuracy, precision, recall and F1-Score. These metrics are derived from the confusion metrics [7], which displays the number of occurrences in which a classifier correctly and incorrectly predicts the outcome of the null hypotheses. In statistics, when there are two different proposed solutions to a problem, the two solutions are called the null hypothesis, or  $H_0$ , and the alternative hypothesis,  $H_1$ . The null hypothesis is the position

of the defendant, or in this case the prediction of correctly classified data by the classifier, while  $H_1$  is the alternative, where data is misclassified [8]. For example, if a data point is predicted to be malicious, the four elements of the confusion matrix would be:

- True Positive (TP): TP represents a correct classification as malicious;
- True Negative (TN): TN represents a correct classification as not malicious;
- False Positive (FP): FP, also referred to as the Type I error, represents an incorrect classification, where the program is identified as malicious while it is benign; and
- False Negative (FN): FN, also referred to as the Type II error, represents an incorrect classification, where the program is identified as benign while it is in fact malicious [7].

Accuracy is the number of true predictions over all the predictions, as shown in Equation 1. Precision is a ratio of correct prediction over all the positive predictions, as shown in Equation 2. Recall is slightly different and is the ratio of the true predictions correctly identified by the algorithm over all the predictions, as shown in Equation 3. It is also called True Positive Rate (TPR). Finally, the F1-Score is a harmonic mean of both precision and recall and is defined by Equation 4. Algorithms that have a high precision and recall also have a high F1-Score [7].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

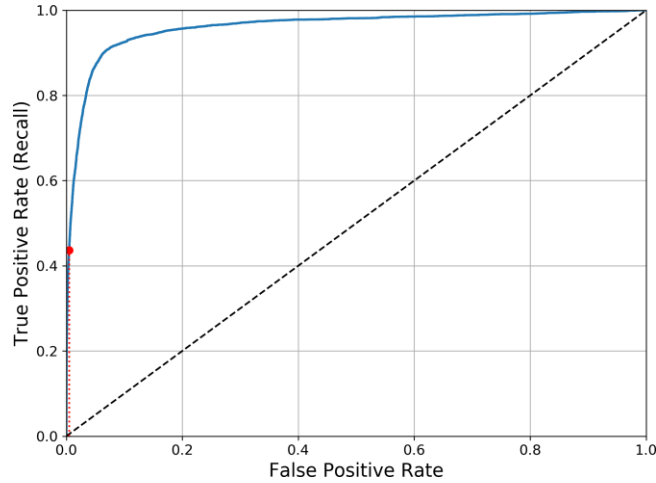
$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (4)$$

Another metric that is sometimes used is the receiver operating characteristic (ROC) curve to visually assess the performance of the algorithm. The area under the curve (AUC) represents the percentage of the graph that is under the curve. The ROC curve is

shown in Figure 1. The ROC curve axis are the TPR, or recall, and the False Positive Rates (FPR), which represent the ratio of the misclassified negative predictions [7].



**Figure 1—ROC curve**

## 2.2 Machine Learning

Machine learning is a field of data science where computers can learn from the data they are provided with. The idea is that the computer, when given a specific task with a performance metric or a threshold, will learn from each experience, or new data, by optimizing parameters in its algorithm so that over time, the task will be achieved more efficiently based on this threshold. Machine learning is beneficial to help solve complex problems that involve significant amounts of data and where the environment is constantly changing [7].

There are two main types of machine learning systems: supervised and unsupervised learning. The main difference between the two types of systems is that in a supervised learning system, the algorithm is trained with a labelled dataset. This means that before it attempts to classify new data, the algorithm knows the desired solutions, such as if a certain behaviour is anomalous or not. Support Vector Machines (SVM), Random Forest (RF) and other decision tree algorithms are examples of supervised learning algorithms. On the other hand, in unsupervised learning, no labels are provided to the

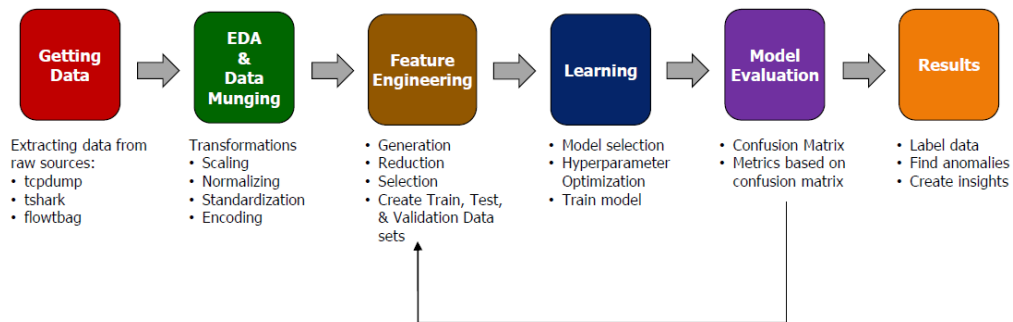
algorithm. It has to make sense of the data on its own. This can include clustering algorithms, anomaly detection algorithms such as Isolation Forest (IF), visualization and dimensionality reduction algorithm such as Principal Component Analysis (PCA) and association rule learning [7]. Figure 2 shows an anomaly detection algorithm that categorized two new data points. Out of those, one was calculated outside of the grouping and therefore deemed anomalous.



Figure 2—Anomaly detection (Reproduced from [7])

Two other types of machine learning algorithms are semi-supervised learning, which is a mix of the two types presented above, where some data is labelled, but mostly it is not, and reinforcement learning, where the algorithm try different actions and get rewards or penalties based on the decisions it takes. At every attempt, the algorithm learn from its experience to maximize the rewards [7].

The machine learning pipeline covers all steps from acquiring data to getting the results. The idea is to collect data, which is in a raw format such as logs, then transform them into something that can be processed by the algorithm, such as numbers. Out of all the features generated, some are more useful than others. A selection of the best features must be done. Then, the model can be trained with the data and see how well it performs. Optimization can be done by tuning one or more of the parameters in the given algorithm. Finally, the model can be applied against new data to classify new data points, or to find anomalies [9]. Figure 3 shows the steps of the machine learning pipeline.



**Figure 3—Machine Learning Pipeline (Reproduced from [9])**

Machine learning can be applied to multiple fields of computer security. Network security research such as Spiekermann and Keller [10] aimed at detecting anomalous network traffic. Host-based analysis research such as Aghaei and Serpen [11] have used machine learning to analyze features collected from hosts, such as system traces. Finally, multiple digital forensic research such as Cohen and Nissim [4], Wang et al. [5] and Mohammad and Alqahtani [12] have used machine learning to automate and assist the detection of malware. Those three areas of research are covered in the following sections.

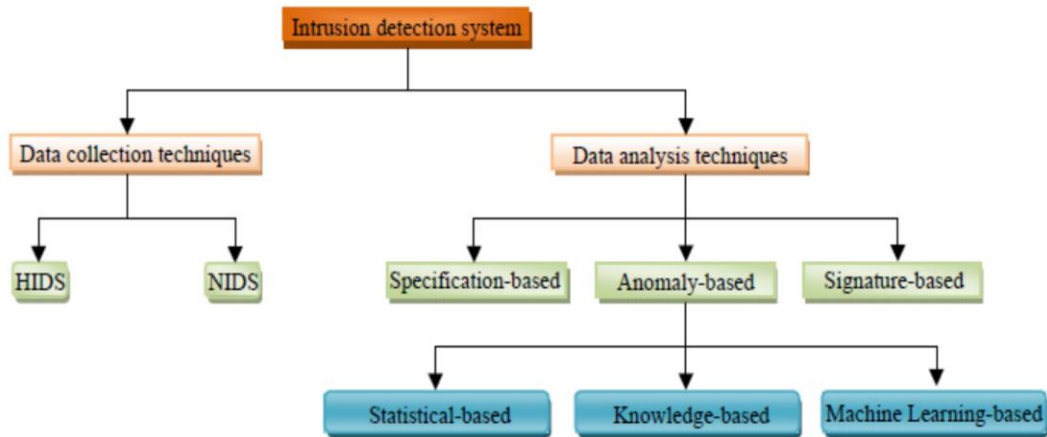
### 2.3 Intrusion Detection System (IDS)

Intrusion Detection Systems (IDS) are tools used to monitor the network traffic and the computers within a network. Their aim is to detect intrusions throughout the steps of the cyber kill chain, covered in Section 2.3.1. The kill chain is a framework to assist in classifying intrusion and analyze them [13].

There are two main types IDS, network-based (NIDS) or host-based (HIDS). Both have a different visibility over the network as they can see different artifacts. This section covers what an intrusion is, and both types of IDS. In addition, it covers the difference between signature-based and anomaly-based detection.

This section covers both types of IDS, as well as the different data analysis techniques, as shown in Figure 4.





**Figure 4—Classifications of intrusion detection systems (Reproduced from [14])**

### 2.3.1 Cyber Kill Chain

Proposed by Lockheed Martin researchers in 2011 [13], the cyber kill chain has become a reference in government and the industry to classify the different phases of an intrusion into a network.

It also proposes applying military intelligence principals to network defence to classify intrusions and share them with allies. As such, they classify the indicators of compromise (IOC) as an important piece of intelligence that needs to be atomic, computed and behavioural. Those IOCs help update IDS rules and prevent future intrusions. They also outline seven common steps in an intrusion. The seven steps of intrusions are [13]:

1. Reconnaissance, where the network is probed for a way in;
2. Weaponization, where malware is generated;
3. Delivery, which is the method used to send the malware to the target network;
4. Exploitation, which is the initial footprint when the payload is run for the first time;
5. Installation, which adds persistence;

6. Command and Control (C2), which sets up a two-way communication with the attacker and enables the attacker to receive and send commands to the malware; and
7. Actions on Objectives, which enables an attacker to exfiltrate data outside of the network or move laterally to increase its presence for future use.

Their kill chain is very linear, but other kill chains such as the Unified Kill Chain from Pols [15] are more fluid. Nonetheless, it is a framework that can be used to share information with other organizations. It is important to note that the earlier in the chain malware is detected, the lesser the impact on a network will be [13].

Figure 5 shows the seven phases for three different intrusions. Note that similar IOCs across the different intrusions can assist in future detection. All three intrusions in Figure 5 use the same installation files and C2 channel. In addition, the first two intrusions have the same encryption key, shellcode and delivery method. The last two have the same delivery mechanism [13]. Those similarities show the importance of effective and rapid sharing of IOCs. They can also help attribute the originator based on the Tactics, Techniques, and Procedures (TTP) used [16].

Phase	Intrusion 1	Intrusion 2	Intrusion 3
Reconnaissance	[Recipient List] Benign PDF	[Recipient List] Benign PDF	[Recipient List] Benign PPT
Weaponization	Trivial encryption algorithm		
	Key 1		Key 2
Delivery	[Email subject] [Email body]	[Email subject] [Email body]	[Email subject] [Email body]
	dn...etto@yahoo.com		ginette.c...@yahoo.com
	60.abc.xyz.215	216.abc.xyz.76	
Exploitation	CVE-2009-0658 [shellcode]		[PPT 0-day] [shellcode]
Installation	C:\...\fssm32.exe C:\...\IEUpd.exe C:\...\NEXPLORE.hlp		
C2	202.abc.xyz.7 [HTTP request]		
Actions on Objectives	N/A	N/A	N/A

Figure 5—Lockheed Martin kill chain (Reproduced from [13])

When developing an IDS or creating new rules, some phases of the kill chain are more useful at detecting malware. For example, using a HIDS, the focus is on the exploitation and installation phases, as this is where the malware executes and generates artifacts on disk and in memory. Understanding what tool and techniques can be used to detect those artifacts is important as it influences what to collect and monitor. In addition, the C2 phase can also be detected due to abnormal processes using network connections. On the other hand, NIDS focus on the delivery and C2 as this is where they have the most visibility. Understanding at which stages of the kill chain malware can be detected and using which artifact is important when developing a rule-based or a behaviour-based detection tool.

### 2.3.2 Network Intrusion Detection Systems (NIDS)

Network intrusion detection has been around since the 1990s and was among the first type of tools available to the first SOCs. Today, due to their FPR and the rising level of encryption in network traffic, due in part to the widespread adoption of HTTPS, there

are increasing concerns over how effective NIDS are becoming. The growing implementation of encrypted Domain Name Server (DNS) traffic only makes this trend more visible. Phishing emails are one of the main entry points into a network, with more than 80% of organizations having experienced such attacks, and 97% of people being unable to recognize a phishing attack [17]. With this type of attack, most of the evidence is visible only on the hosts, especially with encrypted traffic, which makes the NIDS less effective. In this case, the aggregation of logs using a Security Information and Event Management (SIEM) can provide additional information, if tuned properly, and if its database is parsed with powerful software [18]. Machine learning methods discussed in Section 3.1 can assist this process.

Most host-monitoring solutions make use of agents, or sensors, on monitored computers. Those agents share similarities with rootkits, as they need full visibility and enough privilege over the system to have the ability to detect the malware. They communicate with a server so that an analyst can either be alerted to the anomalous activity or query the hosts under observation [18].

Intrusion detection systems usually share common components [19]:

1. data collection from multiple sources and of one or more types;
2. data munging to convert the raw data into usable features for the detection algorithm; and
3. a decision module that determines if an alert is triggered.

On signature-based NIDS, the decision to trigger an alert is based on specific strings or binary patterns matched with a database of malicious signatures. For an anomaly-based detection, a decision function must be implemented. It gives a weight to a given data point and makes a decision based on a predetermined threshold. Machine learning algorithms are helpful as they can make sense of large amount of data to give it a value, or determine outliers using supervised and unsupervised learning algorithms.

### 2.3.3 Host-Based Intrusion Detection Systems (HIDS)

In 1980, Jim Anderson's [20] research was the first to look at how host-based monitoring could be done. At the time, access control was mostly limited to passwords. The interest in monitoring networks was not necessarily focused on malware, but more on insider threat detection. Back in the 1980s, fear of spies and disgruntled employees was the main motivation to secure systems. There was a motivation to start looking at user behaviour, monitoring file access and auditing logs.

From there, the idea of detecting network intrusions and performing network defence started to emerge. Governments started realizing that external threats, in addition to insider threats, were becoming an issue. Works such as the one from Teresa Lunt [21] on audit trails analysis and intrusion detection proposed improvements to Anderson's [20] work and to some other research conducted at that time. Her study of the current research and techniques showed that multiple systems were required to properly monitor a network as each of the analyzed system could be defeated in some ways.

Lunt et al. [22] stated in their research that their Real-Time Intrusion-Detection Expert System (IDES) "monitors the activities of individual users, groups, remote hosts and entire systems, and detects suspected security violations, by both insiders and outsiders, as they occur" [22]. Their IDES was designed to use a mix of rule-based detection, triggering on user-created events, and statistical analysis, trying to define a baseline of a particular user behaviour. They were among the first to propose such system, one that could detect malicious programs and intrusions based on behaviour and signatures.

In recent years, the idea of detecting viruses and intrusions using signatures has evolved significantly. Today, malware, viruses and external attacks are much more common. Consequently, anti-virus software using signature-based detection are widely deployed in modern enterprise networks.

NIDS have been used to report suspicious network traffic for a long time. However, this type of alerting system is also being used to report host-based events, called Host-Based Intrusion Detection Systems (HIDS). Similarly, the deployed agents, which

are programs running locally on the network's hosts, can work together with an HIDS or a SIEM to analyze and report on the current state of the system, which includes procedure calls, registry changes, volatile memory and running programs [19].

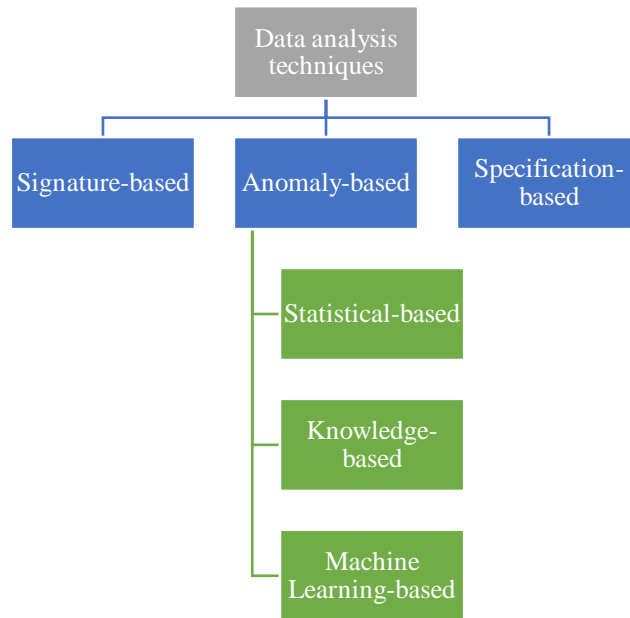
#### 2.3.3.1 *Deployed-Agents*

Deployed agents act both as data collectors and sensors, reporting to the SIEM, with varying levels of autonomy [18]. They can be split into two categories: OS-level sensors, which observe the entirety of the system and look at registries, memory, file changes and program-level sensors which monitor the state of a specific program, likely of importance to an organization [19].

Agents need to be autonomous as remote troubleshooting can be difficult and may affect the end user's productivity. They have to be secured and resilient to attack by being trusted and tamper proof, while being transparent to the users and not adding too much strain on the system. If users cannot operate their system because of the added overhead, the agent is no longer useful.

#### 2.3.4 Data Analysis Techniques

Once the data is collected by the IDS, data analysis needs to be performed to make sense of this data. There are three different data analysis techniques, which are covered in the following sections. Figure 6 shows the different techniques used by IDSs.



**Figure 6—Data analysis techniques (Derived from [14])**

#### 2.3.4.1 *Signature-Based Detection*

Once malware has been analyzed and confirmed as malicious, the best way of preventing other infections from this malicious software in the future is to create a signature that the IDS detection engine then tries to match against the binary of the program, or to its hash value. Signatures can be written to detect malware on multiple systems, such as for NIDS or anti-virus [23].

The main limitation of signatures is its lack of resilience to malware obfuscation, where a single piece of malware can get packed differently or with some added randomness to fool the signatures of the anti-virus engines and the IDS. Malware writers commonly use four methods to avoid detection [24]:

1. dead-code insertion, which is code not affecting the execution;
2. code transposition, which relocates the instructions in the binary, with more jumps to keep the desired behaviour;
3. register reassignment, which uses different registers than what the compiler would normally choose; and

4. instruction substitution, which manually rewrites the compiled code by using similar instructions.

Detection of such obfuscation techniques, especially by automated detection engines using signatures, can be difficult as these techniques involve malware writers with a good understanding of assembly language [24]. In addition, signature-based malware databases need to be updated regularly to ensure the latest detected malware for which a signature exists can be detected [14].

#### 2.3.4.2 *Anomaly-Based Detection*

Anomaly-based detection aims to detect previously unknown attacks for which no signatures are available. As signature-based tools require a signature to see a threat, any new malware or attack vectors, such as zero-day exploits, can go undetected. Anomaly-based detection works by defining what is normal for a given system, and activities that fall outside this baseline are considered anomalous [25].

This baseline, or “learned patterns of normal activity” [14] acts as a line. If an event, a system activity or a network event crosses that threshold, it is considered anomalous. The challenge, however, is to set this line at a point which will be effective at finding true positives, but also generates a manageable number of alerts, which are data points that are flagged as anomalous. If a model can detect every intrusion, but flags half the data points as anomalous, this will generate too many false positive and the anomalies will not be found without significant manual analysis; the IDS is therefore not adding any value. A compromise must be struck to detect as many intrusions as possible while keeping false positives at a manageable level. Adversaries know this as well and will attempt to make their malware look as normal as possible; there malware will be scored much closer to the detection threshold and make it difficult to differentiate from normal activity. To detect them, too many false positives may need to be accepted, and therefore the malware would go through undetected [14].

Once a baseline is established and the rules regarding the system under observation are well understood, anomaly-based detection can scale better than a signature-based



system. Once the baseline is established and the regular business activities are well understood, new intrusions not previously detected and for which no signature exists, can still be detected if it deviates from this baseline [14]. Multiple types of anomaly detection approaches exist, but the following sections will cover four common ones.

#### 2.3.4.2.1 Knowledge-Based Approach

The knowledge-based approach implies prior knowledge of an attack or of the adversary's methodology. Three common types of knowledge-based systems are:

1. the state transition system, where the intrusion is mapped as a state diagram in which the adversary's activity represents the system transition;
2. the expert systems which are ruled-based systems such as implemented by Lunt et al. [22] that takes system logs to which a set of rules are applied, rules that are based on system vulnerabilities, possible intrusion vector and user activity; and
3. signature analysis, which is the analysis of the audit trail generated by the malware and the pattern created as the adversary was operating from within the network [14].

More et al. [26] developed a knowledge-based IDS that can gather logs from multiple sources, including system logs, web-based resources on vulnerabilities and network activities. The information is parsed and considered as facts by the system. The reasoner module looks through those facts to find instances that can be considered as anomalous. To test their system, they ran a remote code execution exploit using a vulnerable version of Acrobat Adobe Reader. The system was able to detect the attack as from its sources, it knew that this version of Adobe Reader was vulnerable by getting the Common Vulnerabilities and Exposures (CVE) feed. CVEs are reports on vulnerabilities maintained by MITRE [27]. The CVE indicated that a malware could use port 80 and that the compromised dynamic link library (DLL) would be "annots.api". Since the same behaviour happened on the system, it detected it.

The advantage of this system is that it leads to accurate results with few false positive. However, the knowledge the system possesses needs to be updated regularly.

#### 2.3.4.2.2 Machine Learning and Data Mining Approach

Data mining is the application of machine learning techniques to analyze a large quantity of data to find new trends [7]. Often covered separately, data mining and machine learning approaches try to make sense of the data and learn from it.

For data mining, this comes under the form of clustering and data classification. Clustering is an unsupervised machine learning algorithm, such as K-means, used to visually separate the data into groups to find new patterns. K-means is an algorithm that allocates all the data points into k number of clusters based on the distance between the points. To do so, an approximative central point is determined for each cluster. Using the arithmetic mean between points, the central point of the cluster is moved so that it is actually centred in the cluster. The calculation is done until the central point of all the clusters stays the same and that each point is allocated to a cluster [28]. Data classification algorithms learn, using a labelled dataset, the distinctive characteristic of each class and attempt to separate them [14]. Laskar et al. [28] proposed an anomaly detection approach that uses Isolation Forest and K-means for real-time anomaly detection using the network traffic logs. Using their approach, they achieved a higher detection rate compared to regular Random Forest implementations, with an AUC score of 0.64 to 0.98, depending on the dataset used to test the implementation.

In machine learning based detection, the model acquires more knowledge about the system as events and data points are collected. The main difference with data mining is that the algorithm tries to learn from the data, not just find new patterns [7] At each iteration, the system is able to better classify the data as it has acquired more knowledge. Three types of machine learning based detection include: Neural Networks, which attempt to anticipate the next step in a way similar to a human brain and feeds the data forward [14], [29]; Fuzzy Logic, which uses approximative truth values that range between zero and one,

using boolean algebra properties [14], [29]; and SVM, which is discussed in Section 3.1.1.3.

Mendonça et al. [30] used Deep Convolutional Neural Network to detect intrusion. Their proposed algorithm achieved better results compared to other current IDS implementation, such as Deep Belief Network, while reducing the processing time. They achieved an F1-Score of 0.97 to 0.98, depending on the type of tested network attacks.

#### 2.3.4.2.3 Statistical-Based Approach

The statistical approach focuses on defining meaningful metrics that can be evaluated statistically, such as by the mean, the variance, the standard deviation and the standard distribution of the data [31]. This can be applied to features such as the interval between user logon or between specific events.

The different statistical parameters can be scored in terms of the deviation from the normal statistical value. If this score is above the predetermined threshold, the event is deemed anomalous. The Markov Model is an implementation of this approach which also incorporates probability and states. [14].

Siris and Papagalou [32] used a statistical-based approach to detect TCP SYN flood attacks. They used an adaptive threshold algorithm and a Cumulative Sum (CUSUM) algorithm to detect the attacks. An adaptive threshold algorithm looks for abnormally high number of packets. A CUSUM algorithm is a change point detection algorithm that looks at the statistical distribution of data before and after a potential change occurs. Based on if the ratio is above a certain predetermined value, the algorithm alerts on the presence of a change. They found that CUSUM was more robust at detecting a wider range of attacks.

#### 2.3.4.3 *Specification-Based Approach*

Specification-based detection focuses on identifying the expected behaviour of a critical system. When an intrusion occurs, the normal behaviour of the system is altered, which indicates that malicious or anomalous activity is ongoing [33]. It is different than an

expert system, described in Section 2.3.4.2.1, as it attempts to build a profile of declarative knowledge, while knowledge-based systems simply use a set of procedures that are known to be normal. This ability to add context to the specifications makes it more flexible and enables the detection of new threats that do not fit the profile.

Liu et al. [34] conducted a survey on potential IDS for smart grids and supervisory control and data acquisition (SCADA) systems and determined that due to the restricted nature of SCADA systems, specification-based IDS would be the best approach to detect intrusions for such system.

## 2.4 Enterprise Network Monitoring Solutions

A lot of open-source and commercial solutions make use of agents to monitor workstations across an enterprise network. Most solutions monitor logs and user activities and report it to the enterprise SIEM or another central reporting server [18]. Tools such as Microsoft Sysmon, Google Rapid Response, Recall, Velociraptor, Encase Enterprise, Carbon Black and many more use agents to feed their central server awareness of all the systems in the enterprise. Those agents are programs that run with elevated privileges. This can also present a risk if malware is able to compromise the agent, as it can hide its presence. Microsoft PowerShell can also be used for this purpose as it can get access almost anywhere on a Windows enterprise network. It can remotely run commands and get the data back to the administrative console. While this requires an adversary to previously have escalated privileges, it can make their actions on objective easier and more covert.

One of the latest tools used by SOC's are EDRs. They are host-based monitoring systems that have agents on the endpoints that report to the central server the collected information, such as Windows events and binaries.

The focus of EDR tools is on the host-based artifacts and not on the network traffic; network information can be gathered, but only from the host logs and files. The main advantage of EDR tools is that they enable real-time collation of data that SOC analysts can monitor [35].

Another advantage of EDRs is that they can act as an Intrusion Prevention System (IPS) and block threats as they are detected. SOC analysts can also use them to quarantine a host and clean the network after an intrusion. This can reduce the time taken to intervene and reduce the scope of the incident [35].

In their assessments of various EDR solutions, Karantzas and Patsakis [35] tested eleven EDRs, including Carbon Black, CrowdStrike Falcon, F-Secure Elements EDR, McAfee Endpoint Protection and Symantec Endpoint Protection, against TTPs used by Advance Persistent Threat (APT) actors. They found that no EDR solution was able to find all the threats. Most of the tested EDRs failed to detect DLL Side-loading, which is covered in Section 3.2.4. [35].

In Figure 7, the ability of the tested EDRs to detect four threats is displayed. First, they tested introducing malware using the Control Panel file extensions (CPL) to hide the malware. CPL files are executables or DLLs registered in the registry key “HKCU\Software\Microsoft\Windows\CurrentVersion\Control Panel\Cpls” [36]. Karantzas and Patsakis [35] used this type of file to introduce shellcode by leveraging memory mapped files, which allows file modifications directly in memory [37]. Second, they used an HTML Application (HTA) file by redirecting a user to a malicious site. The user runs the Visual Basic script (VBS) that injects into mshta.exe. Third, they used an unsigned portable executable (EXE) that performed a process injection and spoofed explorer.exe. Fourth, they used DLL side-loading, discussed in Section 3.2.4, to inject into a signed binary. For each of those attacks, all the eleven EDRs were tested. In Figure 7, each checkmark represents a successful attack. Each full dot represents a successful attack, but where the EDR triggered only a minor alert, which may go unnoticed. Each star represents a successful attack that triggered an alert. An empty dot shows an unsuccessful attack. Finally, a cross shows a failed attack that was detected by the EDR. None on the tested tools were able to detect all four attacks, and a large number of attacks were successful [35].

EDR	CPL	HTA	EXE	DLL
Carbon Black	•	✗	✓	✓
CrowdStrike Falcon	✓	✓	•	✓
ESET PROTECT Enterprise	✗	✗	✓	✓
F-Secure Elements Endpoint Detection and Response	✓	✓	✓	✓
Kaspersky Endpoint Detection and Response	✗	✗	✗	✓
McAfee Endpoint Protection	✗	✗	✓	✓
Sentinel One	✓	✓	✓	✗
Sophos Intercept X with EDR	✗	✗	✓	-
Symantec Endpoint Protection	✓	✗	✓	✓
Trend micro Apex One	✓	○	✓	✓
Windows Defender for Endpoints	*	✗	✗	✓

**Figure 7—Detection results of the various EDRs tested by Karantzas and Patsakis [35] using CPL, HTA, EXE and DLL attacks (Reproduced from [35])**

EDR solutions are adding more and more machine learning algorithms and processing to attempt to increase the detection rate while maintaining low false positives, but also detecting malware earlier in the cyber kill chain, where less damage has been done. The main focus of EDR vendors is to find good features to evaluate while maintaining a low footprint on the host with a minimal bandwidth requirement, as well as efficiently parsing and processing an enormous quantity of data [35].

## 2.5 Memory Forensics

Memory forensics is a newer field of investigation within the digital forensic community. It makes use of tools to analyze the data structures in a system's physical memory and has two main advantages over disk forensic [38]:

- it can find evidence of executables that live only in memory; and
- it can find evidence of rootkits, malicious software that modifies the behaviour of the OS and has full privilege over the system, which enables them to lie to other

host-based detection tools. Live analysis is not reliable as attackers can intercept API calls.

The most common tool for memory forensic is Volatility; it is powerful and has been an industry standard. “The Art of Memory Forensics” [38] goes into a lot of details about its functionalities and intricacies. Being Python-based, it is highly customizable and a lot of researchers have developed new plugins to increase its functionality. Those plugins are available open source and users can easily generate new ones that fit their needs.

Memory forensics is performed using a capture of the RAM of a computer that is about to be shut down to conduct an investigation. A tool such as Dumpit [39] is required to collect the memory of a running computer. Once the computer is powered off, the data in memory is lost and an analyst is no longer able to derive indicators of compromise from the behaviour of the malicious program. Performing the analysis on a live system gives the analyst an opportunity to monitor the system behaviour over a longer period of time and take more captures if required. However, this can be risky if an adversary is exfiltrating data from the computer, as you generally want to shut down the computer as quickly as possible to avoid further damage.

Cohen and Nissim [4] used machine learning and artifacts found in memory using Volatility to detect ransomware in a cloud computing server, hosting hundreds of virtual machines (VM). Using Volatility plugins, such as the Process Cross-View Plugin (psxview), thread scanner (thrdscan), services scanner (svcsan) and list of DLLs (dlllist), they generated features that could be fed into a machine learning algorithm.

The advantage of their approach is that it enables detection of fileless malware, which do not have a presence on the disk. Fileless malware has many forms, including PowerShell code and using Windows Management Instrumentation (WMI) to run without having a presence on the disk [40]. They used VMware’s vSphere infrastructure to collect snapshots of the VMs, then extracted the memory captures from the snapshot files so they could be analyzed using Volatility.

They extracted snapshots every ten minutes, performing basic operations on the VMs, such as running a legitimate program. After 100 snapshots, they had a baseline to compare new data to. Once the baseline was generated, they ran nine programs, four benign (Process Monitor, Wireshark, the Microsoft Windows default Disk Defragmenter utility tool and Avast anti-virus) and five malicious ransomware (Cerber, TeslaCrypt, Vipasana, Chimera and HiddenTear) to examine the effect, taking a new snapshot and reverting to the baseline after each execution.

Figure 8 shows some of the features they generated with the various Volatility plugins. To be processed by their machine learning algorithms, the features needed to be integers that could be weighted and processed properly as most machine learning algorithms are better suited to process numeric values [7].

ID	Feature name	Description	Type	Volatility plugin
1	callbacks_num	Number of kernel callbacks.	Integer	callbacks
2	dlls_dllist_unique_paths_num	Number of unique DLLs used by all processes.	Integer	dlllist
3	dlls_ldrmodules_num	Number of DLLs used by all processes.	Integer	ldrmodules
4	dlls_ldrmodules_unique_mappedpaths_num	Number of unique DLLs used by all processes.	Integer	ldrmodules
5	dlls_ldrmodules_InInit_false_num	Number of DLLs that were not found in the <i>InInit</i> list of process memory.	Integer	ldrmodules
6	dlls_ldrmodules_InLoad_false_num	Number of DLLs that were not found in the <i>InLoad</i> list of process memory.	Integer	ldrmodules
7	dlls_ldrmodules_InMem_false_num	Number of DLLs that were not found in the <i>InMem</i> list of process memory.	Integer	ldrmodules
8	dlls_ldrmodules_all_false_num	Number of DLLs that were not found in the <i>InInit</i> , <i>InLoad</i> , and <i>InMem</i> lists of process memory.	Integer	ldrmodules
9	modules_num	Number of driver objects associated with kernel modules.	Integer	modules
10	processes_privs_enabled_not_default_num	Number of privileges (of all processes), which have the <i>Enable</i> attribute and do not have the <i>Default</i> attribute.	Integer	privs
11	processes_psxview_exited_num	Number of processes that were previously running but were completed before the snapshot was taken.	Integer	psxview
12	processes_psxview_false_columns_num	Number of process listing techniques that do not detect at least one process.	Integer	psxview

**Figure 8—Some features generated by Cohen and Nissim [4] research (Reproduced from [4])**

They used nine machine learning algorithms for their datasets: J48, RF, Naïve Bayes (NB), Bayesian Network (BN), Logistic Regression (LR), LogitBoost (LB), Sequential Minimal Optimization (SMO), Bagging, and AdaBoost (AB). Out of their multiple test cases, RF achieved the best overall results; this has also been observed by Wang et al. [5] who researched kernel-level rootkit using memory forensic and machine learning.

In his work, Hameed [41] used a similar approach, but to analyze a single computer using similar Volatility plugins. The RF algorithm was also used to process the data. Out



of all the plugins used, psxview was deemed to be the most useful. This plugin enumerates processes using seven different methods and compares them to see which method detects each process, using true or false. With the “—apply-rules” option, processes that are not found due to a valid exception, such as terminated processes, are listed as “okay” by psxview [38]. Since psxview looks for hidden processes, this method can only detect rootkits that attempt to hide processes; the usefulness of this method to detect more general rootkits would need to be tested as a hidden process is generally malicious.

```
$ python vol.py -f prolaco.vmem psxview --apply-rules
```

Volatility Foundation Volatility Framework 2.4 (Beta)										
Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcid	csrss	session	deskthrd	ExitTime
0x06499b80	svchost.exe	1148	True	True	True	True	True	True	True	
0x04b5a980	VMwareUser.exe	452	True	True	True	True	True	True	True	
0x05f027e0	alg.exe	216	True	True	True	True	True	True	True	
0x010f7588	waucflt.exe	468	True	True	True	True	True	True	True	
0x04c2b310	wacntfy.exe	888	True	True	True	True	True	True	True	
0x061ef558	svchost.exe	1088	True	True	True	True	True	True	True	
0x06015020	services.exe	676	True	True	True	True	True	True	True	
0x06384230	vmacthlp.exe	844	True	True	True	True	True	True	True	
0x069d5b28	vmtoolsd.exe	1668	True	True	True	True	True	True	True	
0x04a544b0	ImmunityDebugger	1136	True	True	True	True	True	True	True	
0x0655fc88	VMUpgradeHelper	1788	True	True	True	True	True	True	True	
0x06945da0	spoolsv.exe	1432	True	True	True	True	True	True	True	
0x05f47020	lsass.exe	688	True	True	True	True	True	True	True	
0x0113f648	1_doc_RCData_61	1336	False	True	True	True	True	True	True	
0x04a065d0	explofer.exe	1724	True	True	True	True	True	True	True	
0x066f0978	winlogon.exe	632	True	True	True	True	True	True	True	
0x0115b8d8	svchost.exe	856	True	True	True	True	True	True	True	
0x063c5560	svchost.exe	936	True	True	True	True	True	True	True	
0x01122910	svchost.exe	1028	True	True	True	True	True	True	True	
0x04be97e8	VMwareTray.exe	432	True	True	True	True	True	True	True	
0x0211ab28	TPAutoConnSvc.e	1968	True	True	True	True	True	True	True	
0x049c15f8	TPAutoConnect.e	1084	True	True	True	True	True	True	True	
0x05471020	smss.exe	544	True	True	True	True	Okay	Okay	Okay	
0x066f0da0	csrss.exe	608	True	True	True	True	Okay	True	True	
0x01214660	System	4	True	True	True	True	Okay	Okay	Okay	
0x0640ac10	msiexec.exe	1144	Okay	True	Okay	Okay	Okay	Okay	Okay	2010-08-11 16:50:08 UTC+0000
0x005f23a0	rundll32.exe	1260	Okay	True	Okay	Okay	Okay	Okay	Okay	2010-08-11 16:50:42 UTC+0000

Figure 9—Volatility psxview plugin output (Reproduced from [38])

## 2.6 Limitations

Current research has some important limitations that could be improved upon. This section discusses these limitations and how they could be improved.

Signature-based monitoring is limited by the data in its database and is better equipped to detect older threats that have already been analyzed. In addition, the issue with recent research, such both Cohen and Nissim [4] and Hameed [41], is that it is not really applicable to a live environment as the memory images need to be collected through snapshots in order to be analyzed. This does not scale well to recent systems with large amounts of memory. Outside of the cloud-computing environment, disk acquisition and

memory captures need to be done computer by computer, which creates delays and can require a lot of bandwidth if done remotely.

This thesis aims at addressing those factors by collecting at a regular interval digital forensic evidence using Velociraptor. The Velociraptor offline collector, running on each workstation can gather the information about the system state. Features can then be generated on a dedicated server. This enables collecting the artifacts from all computers in the network simultaneously and then process those artifacts using a machine learning algorithm.

Using this technique, it is possible to widen the range of artifacts available to host-based detection tools by looking at volatile and non-volatile digital forensic artifacts.

Generating a baseline helps reduce the FPR generated by the machine learning algorithms. Using those forensic artifacts to monitor the system enable quick response and can lead to the generation of signatures to improve other anti-virus and monitoring solutions. In the case of the implemented methodology, the artifacts collected are directly compared with the baseline using machine learning to identify anomalies.

In this section, a review of current research was presented. It covered different machine learning algorithms and how they work, NIDS and HIDS, enterprise monitoring solutions, memory forensic tools and techniques and limitations of current research. In the next section, further detail on machine learning algorithms, techniques and methodologies are presented. In addition, different types of malware are discussed with some common persistence mechanisms. Finally, Velociraptor, the DFIR tool used in the implemented methodology, is introduced.

## Section 3

# Background

In order to understand the concepts used in this research, additional theory needs to be introduced. This section includes the three algorithms used in this research and how they are optimized, the background theory for the malware used during the experiment and their persistent mechanisms and finally the tool used to collect the data that is processed by the algorithms.

### 3.1 Machine Learning

This section will cover the machine learning techniques and algorithms used during this thesis. In addition, the techniques used to perform feature reduction, to scale and to tune the algorithms.

#### 3.1.1 Algorithms

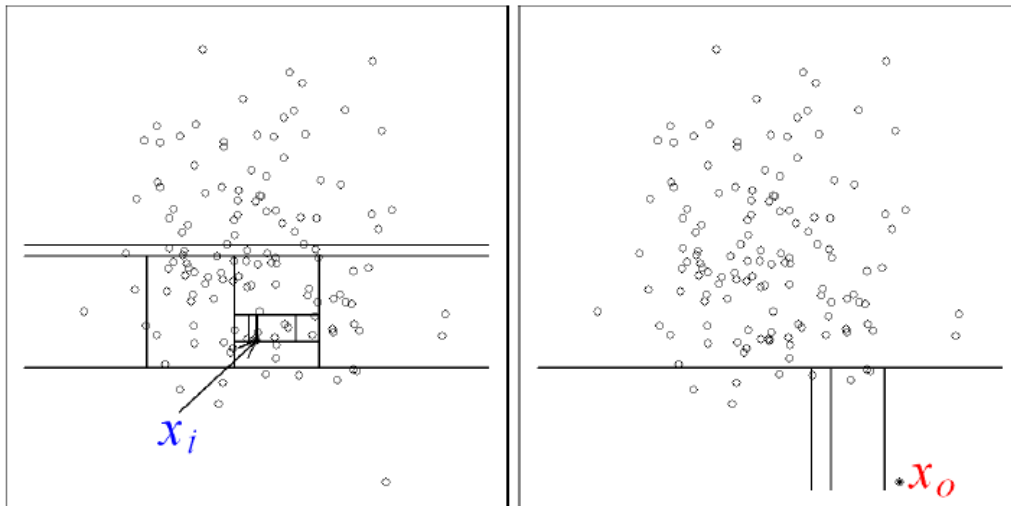
This section covers the three machine learning algorithms used in this thesis, Isolation Forest which is an unsupervised learning algorithm, Random Forest, a supervised learning algorithm and Support Vector Machines, a supervised learning algorithm.

##### 3.1.1.1 *Isolation Forest*

Isolation Forest is an unsupervised learning algorithm specializing in outlier detection [7]. It was first introduced in 2008 by Liu et al. [42]. This technique's main characteristic is that instead of attempting to baseline normal behaviours and then identify what does not fit this defined normality, it focuses on identifying outliers among a large set of data.

Isolation Forest is a tree-based algorithm that splits all the data points into different trees, or splits. This process effectively isolates all data points into different branches of

the tree. The more splits required to reach the data point, the more normal the point is; anomalous data points tend to be easier to isolate and therefore requires fewer splits. This results in fewer branches and makes the process of walking back to the top of the tree shorter. In Figure 10,  $X_i$  is more normal than  $X_0$  for this reason, as the path to the top of the tree for  $X_i$  is longer [42]. The horizontal and vertical lines represent the splits the algorithm made to separate the data.



**Figure 10—Isolating data points using Isolation Forest.  $X_i$  (left) is more normal than  $X_0$  (right) (Reproduced from Liu et al. [42])**

This algorithm has been mostly used in anomaly detection for financial data and for network traffic analysis to find anomalous entries. With Random Forest, outliers are identified as -1 and normal data as 1.

Sharma's [43] thesis analyzed log entries to find anomalous transactions. The dataset used contained credit card information of transactions done over a month period. It contained 284,807 transactions, with 492 of them being fraudulent. The research looked at detecting fraudulent credit card transactions by processing them using Isolation Forest.

Spiekermann and Keller [10], used Isolation Forest to detect anomalous network traffic. Their goal was to detect covert channels, malware usage and other anomalies within

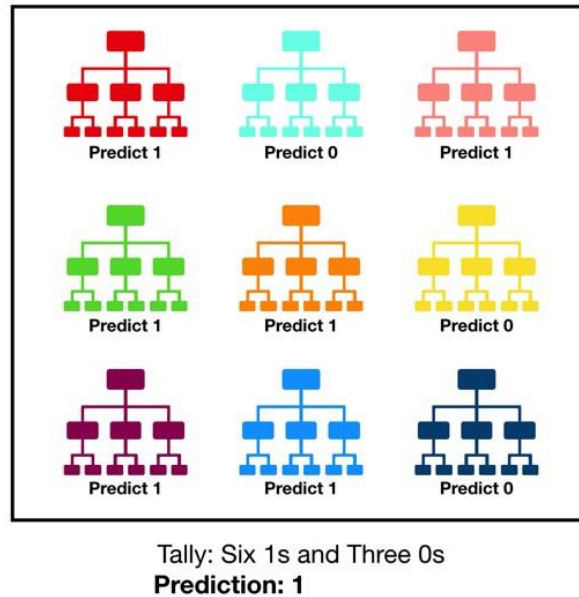
the network. They generated a dataset by performing a packet capture and injecting malicious packets using their test environment.

This methodology can be applied to the analysis of any types of computer logs, such as network traffic or host logs.

#### 3.1.1.2 *Random Forest*

Random Forest was first developed by Ho [44] in 1995. This algorithm is an ensemble of decision trees where multiple trees are built randomly. This makes Random Forest more diverse compared to classical decision tree models. Normally, a decision tree splits a node based on what the best feature is, while Random Forest makes that split based on the best feature of a certain random set of features, to add diversity [7].

A Random Forest prediction is effectively the prediction that is the most occurrent among the overall ensemble of trees. This makes this algorithm outperform other tree-based methods as it reduces the error that a single tree could generate [45]. Figure 11 shows a Random Forest made of nine different trees. Out of these trees, six predicted “1” and three predicted “0”. The Random Forest algorithm predicts the outcome to be “1” since the majority of the trees predicted it. This process is less prone to errors and leads to better performance.



**Figure 11—Random Forest ensemble tree prediction (Reproduced From Yiu [45])**

Random Forest is also a good algorithm to use if the dataset has lots of features, or dimensions, as it is flexible to missing data and can process different types of data, such as continuous, categorical and binary data [46]. Features that use continuous data can have values that range anywhere between two real numbers. For example, distances between two points in metres or the age of a person in years are continuous. Categorical, or qualitative, refers to data that cannot be quantified and can take a limited set of values, such as the name of a city [47]. Finally, binary data can only take two values, such as true or false.

### 3.1.1.3 Support Vector Machine

SVM is a supervised learning algorithms, which aims at classifying the data into a desired number of classes [48]. In the case of this thesis, the SVM separates data into two classes, anomalous and normal data.

SVM is a very good technique to optimize unknown solutions. It can be faster than Neural Network algorithms, but slower than tree base classifiers due to the computational

resources required to select a hyperplane to separate the data. SVM strength is in classifying data in two classes as the hyperplane can be identified to separate data points. It cannot natively support more classes due to the mathematical constraint of the algorithm, though methods do exist for extending or using SVM for multiclass problems. [48].

SVMs works by trying to generate a hyperplane that separates the two different classes of data into separate spaces. In Figure 12, the algorithm found the optimal hyperplane H to separate the data using a linear model. It generated the hyperplanes H1 and H2, which are the first paralleled hyperplanes touching a data point. Equation 5 is used to generate the hyperplanes. H1 and H2 become the support vectors. Of course, this is a perfect case as “hard” margins can be used. Generally, the dataset will not be perfectly separable. In that case, the use of soft margins is required. This is done by introducing a slack variable, which calculates how far from H1 or H2 the point is inside the margin. This is represented by  $\xi_i$ , the non-negative slack variable. Figure 13 shows a similar model as Figure 12, but with soft margins as three of the data points are crossing over H,  $\xi_1$ ,  $\xi_2$  and  $\xi_3$  [48]. To determine if a point will be properly classified, the value of  $\xi_i$  is evaluated. Equation 6 shows the value of the margin. Equation 7 shows a misclassified point, which occurs when  $\xi_i > 1$  and Equation 8 shows a point between the margin and the hyperplane of its class, which occurs when  $0 < \xi_i \leq 1$  [49].

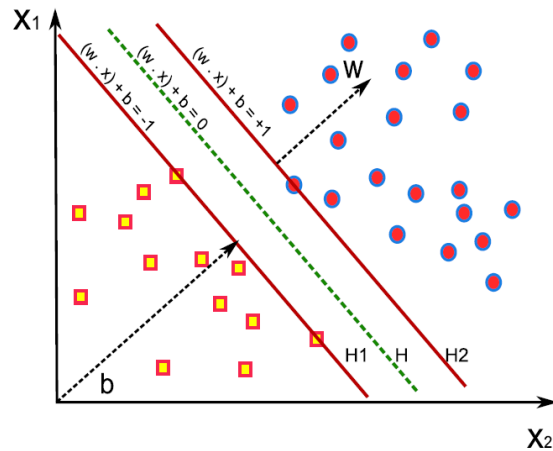
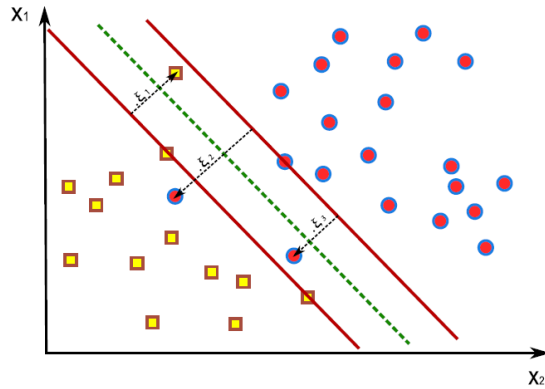


Figure 12—SVM attempting to find the best hyperplane (Reproduced from [48])



**Figure 13—Soft Margin SVM (Reproduced from [48])**

The size of this soft margin can be determined using the parameter  $C$ , the penalty value. This value affects the width of the margin, or the distance between  $H1$  and  $H2$ . The smaller the value of  $C$  is, the larger the margin. This value needs to be tuned during the cross-validation and varies based on the dataset and is discussed in Section 3.1.5.4.

$$f(x) = w^T x + b \quad (5)$$

$$margin = \frac{2}{\|w\|} \quad (6)$$

$$\frac{\xi_i}{\|w\|} > \frac{2}{\|w\|} \quad (7)$$

$$\frac{\xi_i}{\|w\|} < \frac{1}{\|w\|} \quad (8)$$

SVMs can use multiple different kernels to create the hyperplanes. The kernel is the mathematical function of the hyperplane. Equation 5 shows the linear SVM kernel equation, that is used in Figure 12 and Figure 13. However, this is not always optimal. Other non-linear models, such as polynomial and Gaussian Radial Basis (RBF), can also be used to better fit the dataset and create a better separation. RBF is the most common SVM kernel. Figure 14 shows the difference between linear and non-linear SVM. The non-



linear hyperplane, in green, was able to make a perfect separation, while a linear hyperplane, in red, is impossible to separate.

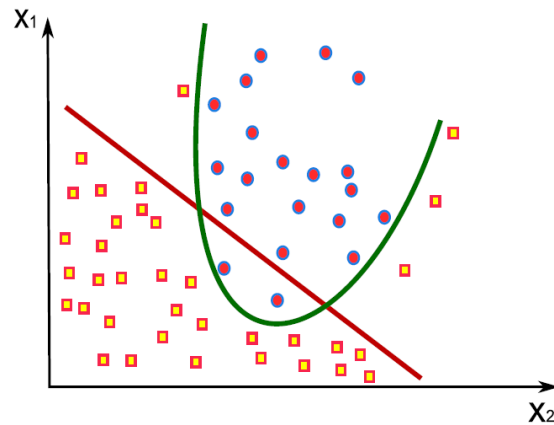


Figure 14—Linear and Non-linear SVM (Reproduced from [48])

Some of the weaknesses to SVMs include long training time on large datasets due to the computational cost, multi-class classifier effectiveness and performance in unbalance datasets [48].

### 3.1.2 Feature Reduction

A dataset can contain numerous features. However, not all features are equivalently useful at classifying the data. The more useful feature improves the model, but some can lead to simply adding more noise. When a dataset contains multiple dimensions, it leads to processing challenges as it becomes computationally heavy to process. Dimensionality reduction algorithms aim at removing the irrelevant and redundant features to improve the model's accuracy [50].

#### 3.1.2.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a feature reduction technique based on linear algebra. It performs a series of orthogonal transformations that aims to preserve most of the variance within a dataset while reducing its dimensionality [50].

The algorithm selects a hyperplane within the dataset that when a projection is applied, most of the variance is preserved. Figure 15 shows a two-dimension example, where three different hyperplanes are considered. On the right, the reduction to one dimension projection is displayed. The top right projection has kept most of the information that was present before and is the best hyperplane. The projection is done on the  $c_1$  axis using the vector  $c_2$  [7].

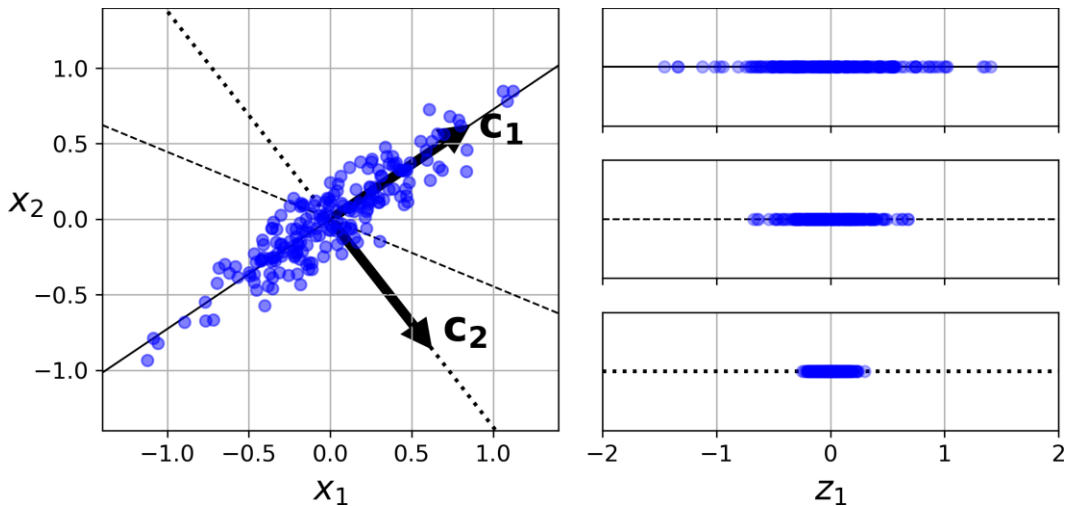


Figure 15—Subspace selection using PCA (Reproduced from [7])

Since with each dimension removed, some information is lost, determining down to how many dimensions a specific dataset can be reduced to in order to retain a large portion of the variance is needed. Usually, the aim is to retain 95% to 99% of the information. To find the optimal number of dimensions, the percentage of variance to be retained can be specified. In Figure 16 shows the SKLearn implementation of PCA, where 95 percent of the variance is retained. Using this, the dataset can be fitted to generate the reduced dataset.

```
pca = PCA (n_components=0.95)
X_reduced = pca.fit_transform (X_train)
```

Figure 16—PCA reduction to retain 95% of the variation, using the SKLearn library

### 3.1.3 Feature Selection

Supervised feature selection techniques can be used to identify which features are more important and should be retained to train the machine learning model. This section covers the SKLearn *feature\_importances\_* variable, the Correlation matrix and Predictive Power Score (PPS) matrix.

#### 3.1.3.1 Feature Importance

Feature importance can be measured using tree-based algorithms, such as Random Forest. Using SKLearn, after a model is fitted with an ExtraTreesClassifier classifier, which works similarly to Random Forest but uses randomized decision trees [51], the variable *feature\_importances\_* can be accessed. Each of the features is attributed a score, which is a weighted average of how much influence a given feature can have on the purity of the tree, across all trees of the forest. The higher the score, the more useful the feature is at predicting the model [7].

#### 3.1.3.2 Correlation Matrix and Predictive Power Score (PPS)

The correlation matrix is a table that shows the correlation coefficient of the features in the dataset. Looking at this table helps understanding the dataset and the relations within its data. It also helps remove features that are less useful and which in turn reduces the computation time to process it [52].

The correlation matrix values range from -1 to 1. A strong positive correlation exists between features the closer the value is to 1, and a strong negative correlation exists the closer the value is to -1. A value close to 0 indicates a limited or non-linear correlation between the data [7].

The Predictive Power Score (PPS) is a new algorithm introduced in 2020 [53] to explore the data and find relationships between the different features. It looks at the probability that any given column, which is one of the features of the dataset, can predict the next column [54]. Using PPS, features that brings the most predictability in the model can be selected. The PPS score works in a similar fashion as the correlation score.

PPS was designed as an improvement over the correlation matrix to see how the result of a column can help predict another column. PPS is also asymmetric, meaning that if X can predict Y, Y does not necessarily predict X. In the correlation matrix, it is symmetric, but values can be negative. This is shown in Figure 17, where we have on the left the correlation matrix, and on the right, the PPS matrix for the Titanic dataset [55]. The colour coding also helps identify the correlation as a darker colour indicates a higher correlation. From this, the most important factor for survival can be identified, which was the sex of the passenger as it has the highest PPS score of 0.57 [54].

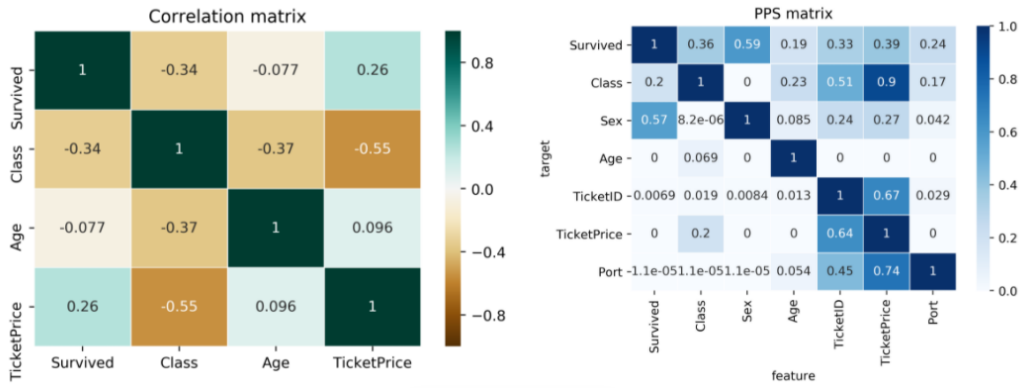


Figure 17—Correlation matrix vs. PPS matrix (Reproduced from [54])

From the PPS, it is possible to perform feature selection by keeping only the features of a dataset where the PPS score for the row containing the label is not zero.

### 3.1.4 Scaling

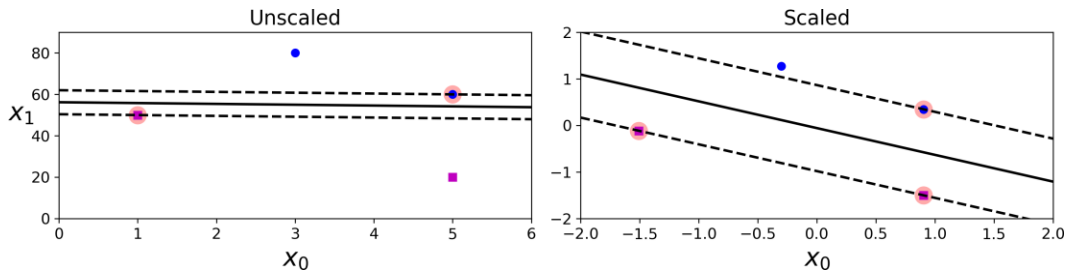
Scaling is important to normalize the data. Most machine learning algorithms have poor performance when the range of the value of the features is not scaled. For example, if one feature in the dataset has its values ranging from 0 to 1 and the other from 0 to 1000, the algorithm could attribute more weight to the latter as the values are bigger. To fix this, two common methods are min-max scaling, or normalization, and standardization. Min-max scaling converts the values of all the features in the dataset so that they range between

0 and 1 by using Equation 9. Standardization achieve a similar result by using the standard deviation and the mean value, as shown in Equation 10 [7].

$$\text{Normalized value} = \frac{\text{Value} - \text{min\_value}}{\text{max\_value} - \text{min\_value}} \quad (9)$$

$$\text{Standardized value} = \frac{\text{Value} - \text{mean\_value}}{\text{standard\_deviation}} \quad (10)$$

Some algorithms require scaling more than others such as SVM. Figure 18 illustrates this. The separation of the data points by the hyperplane is clearer when using a normalized dataset.



**Figure 18—SVM sensitivity to scaling (Reproduced from [7])**

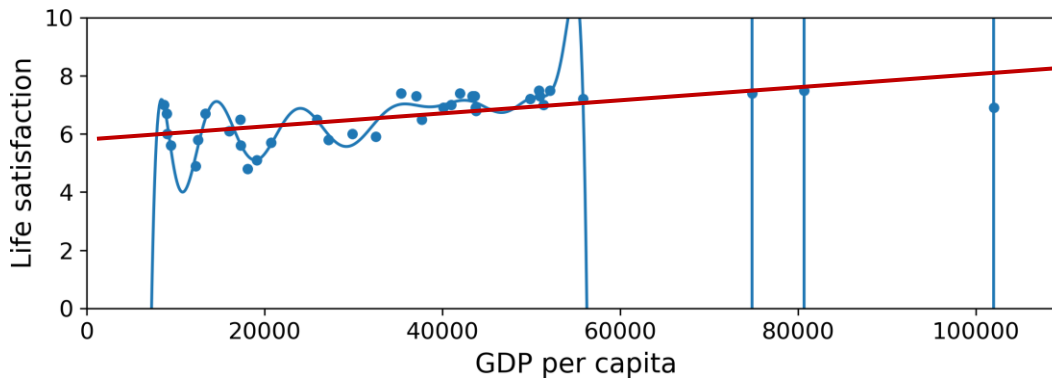
### 3.1.5 Parameter tuning

In order to optimize the results of the different machine learning algorithms, care must be taken to avoid overfitting. This section covers overfitting, as well as multiple ways to reduce it and get better performance out of machine learning algorithms.

#### 3.1.5.1 Overfitting

Tree algorithms have a tendency to overfit the data on which they were trained on. This happens when the algorithm is tailored too perfectly to the training set and has difficulties adapting to the test set. If both are very similar, it can have difficulty identifying new data in the future [56]. For example, in Figure 19, the data point in blue are mostly aligned along the red line. However, an overfitted model could try to adjust too closely to the training set, as shown by the blue line. In the test set, if any other data points are

perfectly aligned on the red line, which represents a model more appropriately tuned, but happen not to be on the overfitted line, they would be considered as outliers.



**Figure 19—Example of overfitted data (Reproduced from [7])**

In his study of overfitting and its solutions, Ying [56] explains the three main reasons why overfitting may occur. First, if there is too much noise, or less representative data, this can lead in the relevant data being suppressed. Second, if a model is tuned too much for high accuracy, it will be less consistent, as aiming for high accuracy in training can lead to higher bias, as it was the case in Figure 19. Lastly, when optimization is performed by comparing feature performance, the feature with the highest score gets selected. However, as some of the features are less useful, the overall accuracy of the model can be reduced as variation in those superfluous features will impact the overall accuracy.

Solutions to overfitting include the use of a separate test and train dataset, as well as cross-validation [57]. In addition, increasing the number of data samples, using fewer features, using early stopping, which means stopping the model while new training iterations improve the model, using regularization to make the model simpler or using ensembling to combine different model predictions to get a more accurate picture, can also help reduce overfitting. Two common ensembling techniques are bagging and boosting [58].

### 3.1.5.1.1 Bagging

Bagging, or bootstrap aggregating, is a technique that uses a single machine learning algorithm and trains on multiple samples that are generated randomly from the training data set. The algorithm will classify each sample and work as an ensemble to select the subset which provides the best performance. Each sample and classifier can work in parallel, which makes the computation faster. Figure 20 shows a training set split into four samples. Each sample is used to classify the data and the ensemble will determine the overall prediction of the model. Using this technique does not necessarily reduce the bias, but provides more consistency by reducing the variance [7].

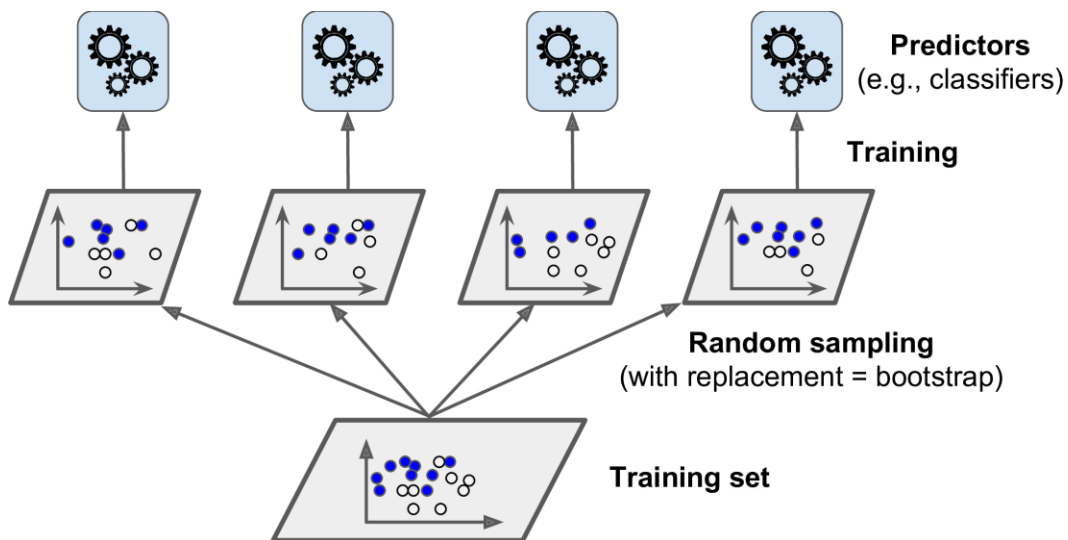


Figure 20—Bagging used to train a machine learning model (Reproduced from [7])

### 3.1.5.1.2 Boosting

Boosting is another ensemble technique that combines multiple strong and weak learners together. However, compared to bagging, it is run sequentially, where each classifier attempts to classify the data and tries to improve the prediction made by the previous classifiers. A common boosting method is AdaBoost, which is a boosting algorithm that focuses on improving from the previous classifiers by focusing on underfitted data. For example, in Figure 21, the AdaBoost algorithm attempts to classify a small dataset. The first attempt focuses on all the points and provides an initial

classification, shown as the red line. For the second attempt, the weights are adjusted to focus on the underfitted points, and this process continues for the number of iterations desired [7].

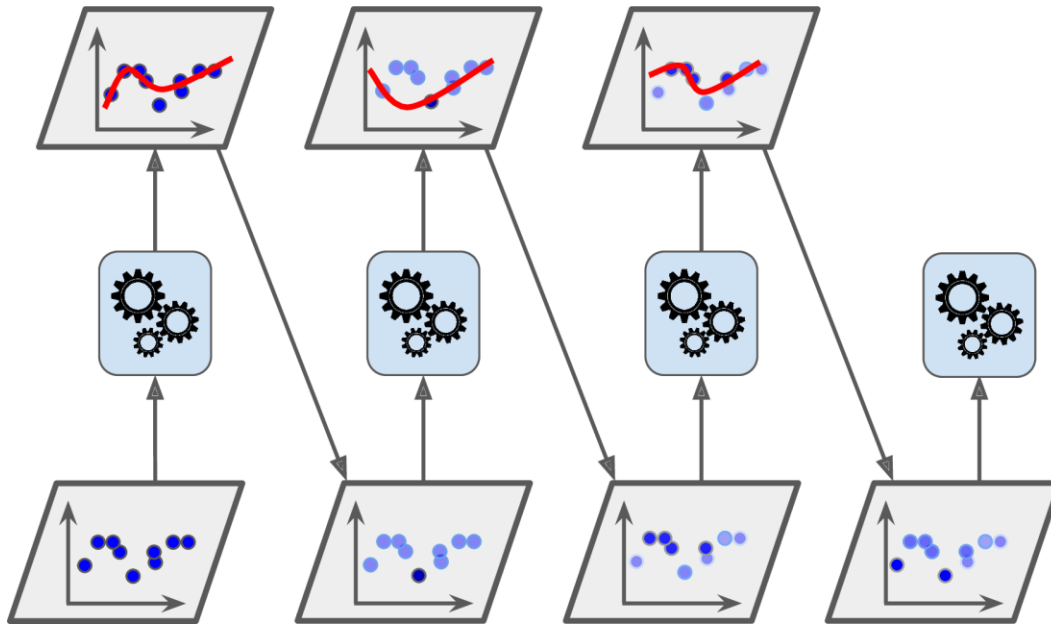


Figure 21—AdaBoost, a common boosting algorithm (Reproduced from [7])

### 3.1.5.2 Grid Search

One of the main challenges in machine learning is the tuning of an algorithm various hyperparameters. Machine learning algorithms have two types of parameters: the parameters, which are determined automatically during the training of an algorithm and the hyperparameters, which need to be provided to the train method during the training of the algorithms [59]. If no values are specified, default values are specified by SKLearn. Those hyperparameters vary for each algorithm, but includes, for Random Forest, the number of estimators, the maximum depth of the tree and the maximum number of features to consider for an iteration [60]. Depending on the hyperparameters values, the performance and the accuracy of a model will vary greatly. Grid Search is a systematic solution to this problem.

The goal of Grid Search is to identify, out of the different possible hyperparameters of an algorithm, the values that will lead to the best prediction while minimizing



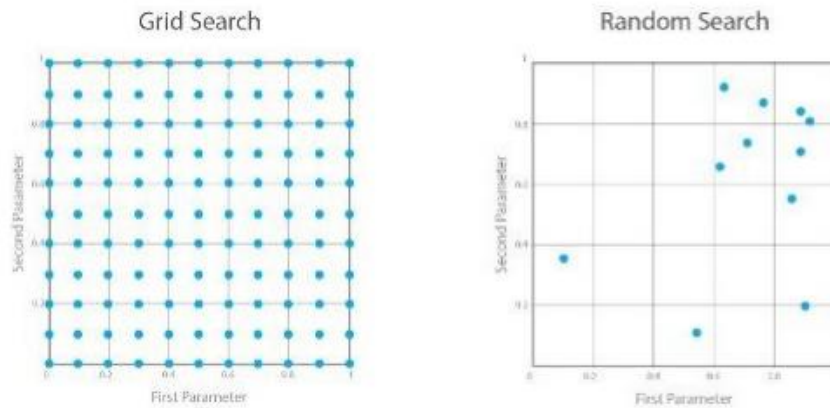
overfitting. A range of values is specified to the algorithm, for each hyperparameters, which forms a grid, and each value is exhaustively tested [61].

This makes Grid Search a computationally heavy algorithm as the more dimensions it has, the longer it takes to compute [62]. This means that the numbers of hyperparameters tested and the range at which those will be tested need to be limited. Therefore, for each algorithm, the most important features can be tested against a limited range of values.

### 3.1.5.3 *Random Search*

Another option is to perform a random search or a manual search prior to the Grid Search to narrow down the parameters and the range at which the performance will increase.

Compared to Grid Search, Random Search only tries some of the combination present in the grid, which leads to faster result. While not perfect, it can help narrow down which hyperparameters to tune and the range of the values [61]. Figure 22 shows the difference between Grid Search, on the left, and Random Search, on the right. Each dot represents a tested combination. Random Search performed fewer tests, but much faster.



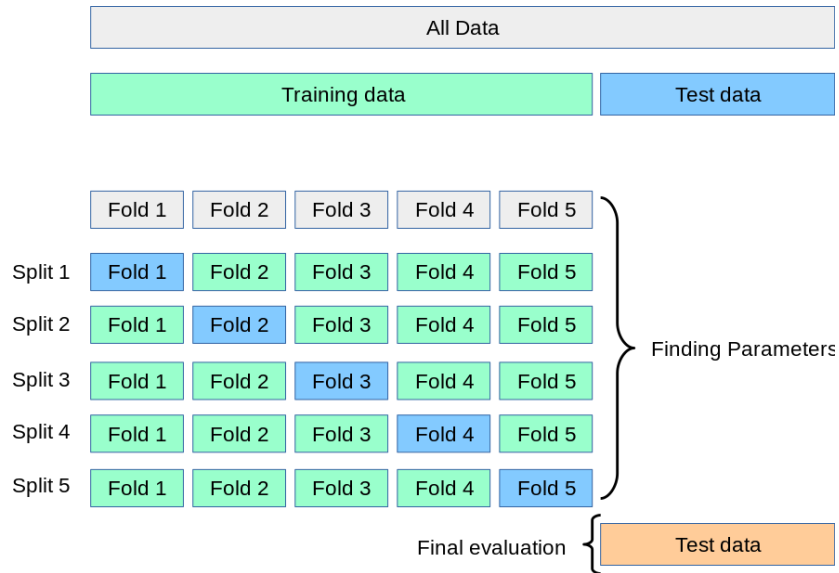
**Figure 22—Comparison of Grid Search and Random Search (Reproduced from [61])**

#### 3.1.5.4 *Cross-Validation*

Once the data is collected, it needs to be separated into two categories to validate the results: a training set and a test set. This is to ensure that the model can be tuned to correctly predict the training dataset, but be flexible enough to predict new data, the test set, accurately [63]. The advantage of doing so is that once you test your model, the data provided to the model has never been seen before. Multiple methods exist to do this, varying in complexity and performance.

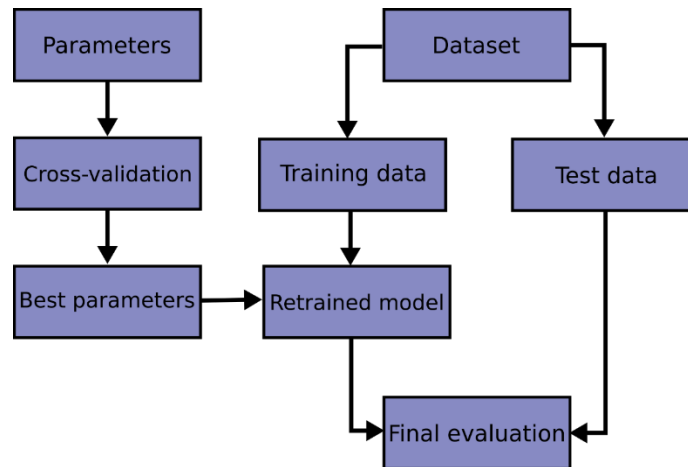
The simplest and computationally simple method is to do a random split of the data into the two categories: the training set and the test set. Using Scikit-Learn, this can be achieved using the “train\_test\_split” function to separate the data. The ratio used is usually between 50% to 75% of the data in the training set and the rest in the test set [7]. However the disadvantage of such technique is that if the dataset is small, only a limited quantity of data is left for the test set, which can impact the results [64].

In addition, if the training dataset is used to tune the model, it can generate overfitting. To fix this, holdout validation can be used [7]. This uses a third dataset, called the validation set. However, if the available data is limited, this can be an issue as more of the data needs to be set aside and cannot be used for the test. Cross-validation methods such as k-fold cross-validation can help fix this issue without having to generate a validation dataset [65]. Usually, the optimal value of k is between five and ten splits [65] [66]. The k-fold algorithm works by separating the whole dataset in k splits, or folds. It then trains the model with k-1 folds and test it with the reserved sample. This process is repeated k times, by rotating which sample is used to test. Once this is done, k different scores have been generated and can help identify the standard variation between each score to see how well the model generalize. If the variation is low, the model is more precise [7]. Figure 23 shows a representation of a fivefold cross-validation. In this example, the training data is split into five components. At every iteration, a different fold is selected to serve as the test data. Depending on the specific k-fold algorithms, this can be stratified so that the labels in the dataset are split equally in the different subsets. This is called stratified k-fold.



**Figure 23—Example of a 5-fold Cross-Validation (Reproduced from [67])**

Using these techniques, an un-bias tuning of the hyperparameters, using the Random Search or Grid Search algorithms, can be performed. The new optimal hyperparameters can finally be used to fit the training set, and then on the test set to get the final results. Figure 24 shows this complete process.



**Figure 24—Cross-Validation flowchart (Reproduced from [67])**

The different machine learning methods and techniques presented in this section covers the six steps of the machine learning pipeline, presented in Figure 3. These methods are used to enable effective use of machine learning to classify intrusions or malware. However, in order to initially have relevant data to work with, during the data gathering stage, malware needs to be executed so it can be labelled as such. Therefore, it is important to understand how the different types of malware operate so relevant features can be selected and fed to the algorithms.

## 3.2 Malware

In the previous sections, multiple techniques aimed at detecting malware have been presented. Such research needs malware samples to train the machine learning models and test their effectiveness. As such, a better understanding of the different types of malware and their capabilities is required.

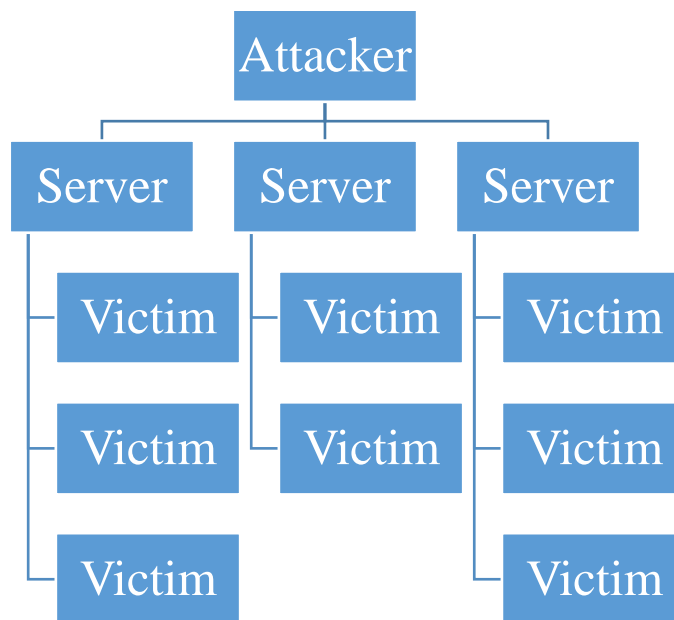
Multiple types of malware exist, some focus on stealing data while others attempt to remain on the infected computer for a long time. In this section, the different types of malware used to generate the malicious samples of this thesis are discussed, as well as the theory behind them.

### 3.2.1 Remote Access Trojan

The Remote Access Trojan (RAT), or Remote Access Tool, is a type of trojan, which is a seemingly harmless program that has a hidden malicious purpose, and is controlled remotely. A RAT enables an attacker to remotely access and control the computer.

The client side, or victim side, usually takes the form a file that has been modified with some shellcode. It can be introduced through phishing using a malicious link or malicious attachments. The server side is an application that intercepts the initial beacon sent by the client over multiple protocols, such as TCP or UDP [68].

The objective of a RAT is to get a foothold into a network and perform specific actions, such as moving laterally to exfiltrate important data, or to enable an attacker to spy on the user by monitoring his activity using screen captures or keyloggers for example. An attacker generally uses multiple servers to control multiple hosts [23]. Figure 25 shows an example of a multi-server/multi-victim communications structure that can be used with RATs. The attacker can have multiple servers to which the victim computer, infected with a RAT, reports to.



**Figure 25—RAT Structure**

Post Exploitation Tools are using RATs to gain and maintain access to a remote computer. Meterpreter, from the Metasploit framework [69] is a commonly used tool that can be used to create and maintain C2 with the RAT. Red teams use it to perform penetration testing [70]. Meterpreter also makes it easy to backdoor executables [71].

### 3.2.2 Rootkit

A rootkit is a program that has the ability to hide the presence of malware on the system. This is achieved by gaining enough privilege on a computer to hide files, processes

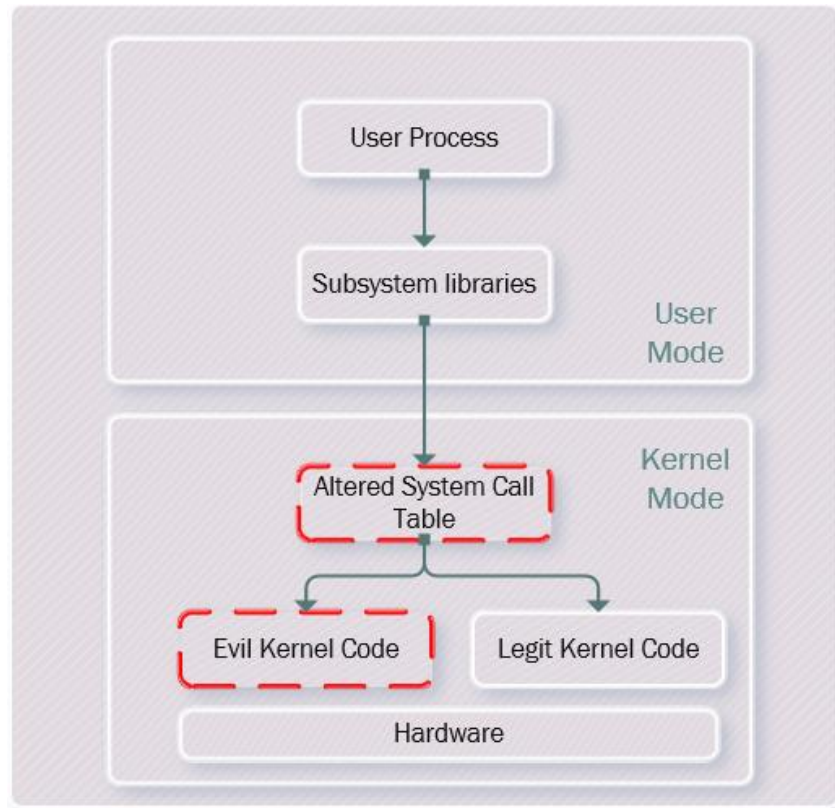
and network activity. This is generally done by modifying the kernel of an OS, such as hooking the API calls with a kernel-mode or user-mode rootkit [23].

Rootkits are a form of backdoor into a system. They attempt to hide the execution of other malware or the attacker's activity on the infected host. There are two main types of rootkits: user-level and kernel-level. The user-mode rootkit is more limited as it operates with user privilege. DLL injection is a common form of user-level rootkit [72].

A common process injection method for rootkit is called process hollowing. Process hollowing happens when the injected executable overwrites another process in memory with its own malicious code. Its goal is that when the malware executes, the process listing tools lists it as the original legitimate binary, such as "svchost.exe" [38].

Kernel-level rootkits are much more dangerous as they can have complete control over the system and hide data much more easily, due to running at the lowest level of the OS, with the most privilege; they are therefore much harder to detect [73]. They do not give administrative privilege, this must be done before through other privilege escalation means [74].

The System Call Table (SCT) is a table in the OS that points to the different system calls. Normally, those pointers should lead to areas within the kernel static code. In case of a rootkit, it can point to other areas of the kernel memory [38]. In their survey of rootkits, Joy et al. [73] describe three techniques a kernel rootkit can use to modify the SCT. Those are the SCT modification, System Call Target modification and SCT redirection. An SCT modification occurs when a system call is modified with a malicious version by changing the address of the pointer, as shown in Figure 26. In that example, the altered SCT now points to a malicious code instead of the normal version. A System Call Target modification occurs when the legitimate destination code is modified by adding a jump instruction that leads to the malicious code. Finally, an SCT redirection occurs by overwriting the SCT in memory with a malicious version.



**Figure 26—Kernel-level rootkit implementation (Reproduced from [74])**

To detect rootkits, a memory forensic analysis of an infected computer using Volatility is one of the best options. Plugins such as psxview discussed in Section 2.5 can detect rootkits as it lists processes on the host using seven techniques. It is unlikely the rootkit will be able to hide from all of them. In addition, the plugin apihooks and driverirp can help detect hooks on the system or rogue drivers [38]. Those techniques have been leveraged in some recent research, including Wang et al. [5] that used Volatility plugins and machine learning to detect kernel-level rootkits. Their method led to an accuracy of 0.986, an FPR of 0.076 and an AUC score of 0.998 using the RF algorithm.

Since Windows Vista, Microsoft requires that kernel-level drivers are digitally signed in order to be loaded on the computer [75]. This makes kernel-mode rootkits rare today.

### 3.2.3 Ransomware

Ransomware is a currently popular type of malware mainly used for financial gain by cyber-criminals. There are two main types of ransomware: Locker and crypto—ransomware.

Locker ransomware locks out the user of the computer and displays a message with the instructions to unlock the computer. This type of ransomware usually blocks access to the computer graphical user interface (GUI), but leaves files intact. It can be cleaned more easily [76].

The more powerful form of ransomware is the crypto ransomware. This type of ransomware encrypts the user files on the computer. System files are left intact so that the OS can remain stable so the user can pay the ransom. The malware informs the user of the payment options to get the decryption key that will restore the original files. Due to the nature of cryptographic algorithms, it is almost impossible to retrieve the files without having a backup of the files somewhere else. Since it only encrypts user files, it does not require elevated privilege [4].

Cohen and Nissim [4] focused on detecting ransomware using Volatility plugins and machine learning, as presented in Section 2.5. They obtained good results using Random Forest, with an AUC score of 0.966 and a F1-Score of 0.935 on average across their experiments.

### 3.2.4 Fileless Malware

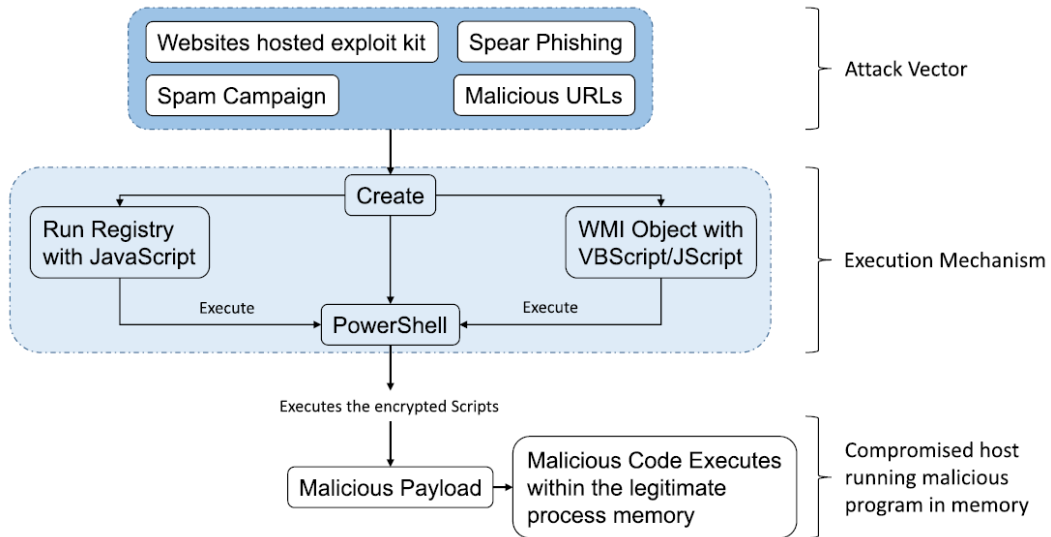
Fileless malware is a type of malware that does not have a presence on the disk. Fileless malware are also called Windows living off the land binaries (LOLBins) as they use tools already present on a host to perform their actions. LOLBins are more common nowadays since forensic investigators generally look for malicious files introduced onto the computer. They provide a more stealthy way of infecting and adding persistence into a host [77]. In their study of fileless malware, Sudhakar and Kumar [77] discussed the use of legitimate software present on user workstations, such as flash player, the web browsers, PDF viewers and Microsoft Office in such attack. Fileless malware can be used to run



scripts that execute in memory and have no presence on the disk; it can take the form of a web exploit, the use of macro-embedded documents and other scripts. Two tools that are integrated into the Windows OS and commonly used by attackers are Windows Management Instrumentation (WMI) and PowerShell. Using those tools, an attacker does not have to bring additional software into the network [77].

During a fileless attack, the flow of execution is as presented in Figure 27. The first step is the foothold, which can be established through various means, such as an infected link or a phishing campaign. Then, from this initial access, the adversary establishes persistence. This can be done in many ways, including adding registry keys. Finally, PowerShell is the weapon of choice for an adversary as it makes it a lot easier to execute programs without any presence on disk and is a very powerful tool [77].

Other tools such as “PsExec.exe” are often used. However, it is not part of the core OS as WMI and PowerShell are, in many cases use of “PsExec.exe” would have to be introduced into the target environment [78].



**Figure 27—Infection flow of files malware (Reproduced from [77])**

Fileless malware can take many forms. Sudhakar and Kumar [77] classified them in three categories based on their persistence mechanism: memory resident malware,

Windows registry malware and rootkits fileless malware. To detect such malware, they recommended either monitoring the system behaviour, such as Windows events ID 4688 (new process has been created) for a new PowerShell process, 7040 (new auto-start services) and 10148 (listen to specific IP and ports). Additionally, they recommended using rules to detect such malware. For example, monitoring instances of the Windows command prompt or of PowerShell that have been launched from a Microsoft Office document [77].

Afreen et al. [40] acknowledge that little academic research has been performed on this type of malware due to their volatile nature. Security companies do most of the research on this, but with limited publishing.

### 3.3 Persistence Mechanisms

Once a foothold has been established in a network, malware authors want to make sure they can maintain their access once the computer restarts by establishing a persistent backdoor [79]. Dubey [80] lists multiple different malware persistence techniques, and are shown in Figure 28.

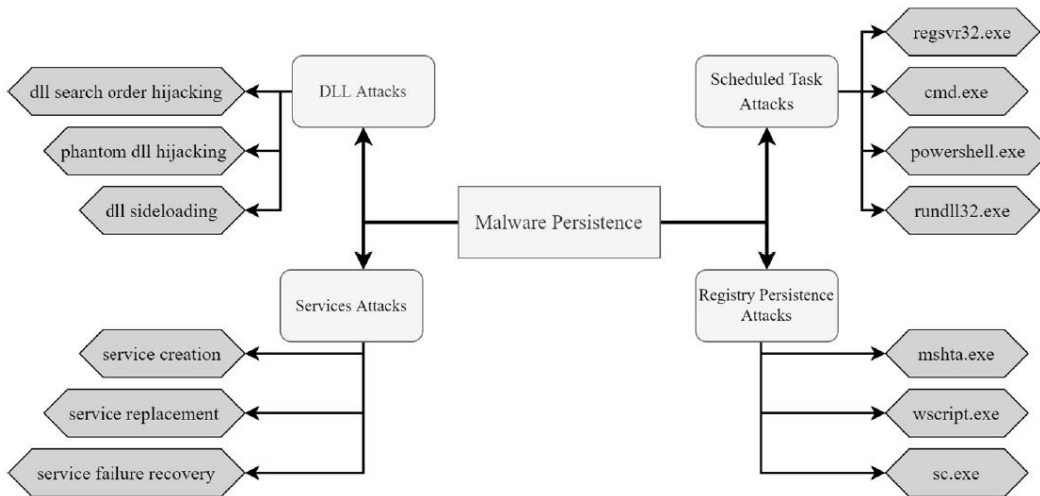


Figure 28—Various persistence mechanisms used by malware (Reproduced from [79])

Rootkits and bootkits can be used to establish persistence. This can be done by modifying parts of the startup code of the computer that is run during the boot sequence. This can be achieved by modifying the Master Boot Record (MBR), Volume Boot Record (VBR) or a boot sector so that the malware can be started during the boot sequence [80].

Registry key modifications can also establish persistence. A common example is to add or edit a registry key so that the malware is run automatically at startup, as shown in Figure 29 if the attacker has user-level privilege and Figure 30 if he has administrator privilege [80].

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
```

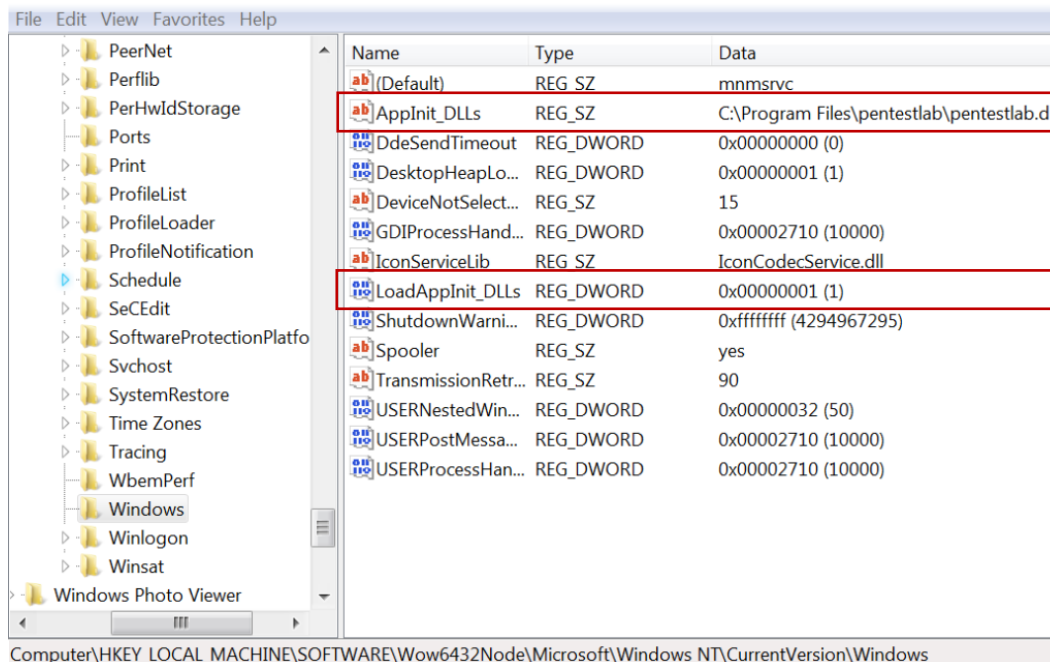
**Figure 29—User-level persistence registry keys**

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\
Run
```

**Figure 30—Administrator-level persistence registry keys**

Similarly, by adding the malware to the folder “C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup”, the program is executed at user logon [80].

A common DLL used for persistence is the “AppInit\_DLLs”. By default, this is disabled by Windows for security reasons. However, it can be enabled by setting the value of the registry key “LoadAppInit\_DLLs” to “1” and by modifying the path of the DLL that will be executed, the “AppInit\_DLLs” registry key. This way, malicious DLLs can be added to every new process. This is shown in Figure 31, where the malicious DLL “pentestlab.dll” has been set as the “AppInit\_DLLs” key value. In this case, it is a Meterpreter beacon, so every new process on the infected host creates a new Meterpreter session on the attacker’s computer [81].



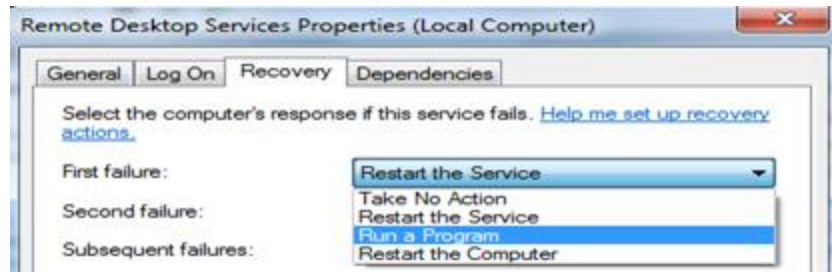
**Figure 31—AppInit\_DLL Registry key (Reproduced from [81])**

DLL manipulation is also a very good persistence mechanism. For example, due to the DLL search order on a Windows computer, when a DLL is requested by an executable, Windows first looks for it in the executable’s directory. This can also be modified using registry keys [82]. If a malicious DLL has the same name as the legitimate one, but is higher in the load order, it is loaded instead. This is also called DLL Side-loading [83]. Similarly, a malicious DLL can be hooked to a process, meaning that when the process starts, it loads the malicious DLL as well.

Malware can also make use of the Windows Task Scheduler to add persistence. Using this tool, they can set the malware to start at startup, but also at multiple different schedules. Scheduled tasks can run PowerShell scripts or programs, which can reach back to the attacker [79].

Windows Services can also be leveraged. As shown in Figure 29 and Figure 30, services can be run at startup. If a rogue service is created and added to the list, it creates persistence. In addition, if a service fails to start, the user has the option to select some

recovery actions. This includes running a different program. The adversary can ensure the service fails to start and add a recovery action to run the malware, as shown in Figure 32 [84].



**Figure 32—Actions if a service fails to start (Reproduced from [84])**

In summary, there are multiple techniques that can be used to establish persistence on a computer. Whether it is by using DLLs, Scheduled Tasks, Windows Services or the Registry, attackers have multiple options to stay on the compromised workstation [79]. Figure 28 shows different types of persistence mechanisms.

### 3.4 Velociraptor

Velociraptor is an endpoint monitoring and Digital Forensics, Incident Response (DFIR) tool built to collect forensic evidence, monitor events, facilitate the enterprise threat hunting efforts and respond to an incident in an enterprise network. This project is very recent (2019) and is actively maintained. It has hundreds of plugins and can be extended by writing new ones that are tailored to an analyst's needs [85].

Velociraptor uses query language similar to the Structured Query Language (SQL), the Velociraptor Query Language (VQL), to analyze the data directly. It allows parsing the network computers as if they were elements of a database. It also offers analysts a notebook to write their observations and run VQL queries, and also supports the exportation of data into a Jupyter notebook [86]; Jupyter makes it easier to write reports, generate graphs and use machine learning algorithms to parse a large number of data [87].

Velociraptor has four main functions: collect artifacts, monitor a system, hunt for a specific threat and respond to an incident. During a collection, the processing is done by the agents on the host under observation, and only the results of the queries are sent over the network, saving significant bandwidth. It is lightweight and transparent to the user. The monitor function displays real-time events, such as DNS requests, event logs, file modifications, process execution and Event Tracing for Windows (ETW) architecture and display them on the Velociraptor server as they occur. ETW is a Microsoft Windows kernel-level tracing mechanism that enables tracing of kernel-mode or application-mode events to a log file [88]. The hunt function is where an analyst actively looks for malware or suspicious activity, generally based on threat intelligence or a currently known vulnerability [85].

When doing a collection, Velociraptor artifacts look very similar to Volatility's artifacts; Volatility is a widely used memory forensic tool, which is covered in Section 2.5.

Velociraptor has limited memory forensic analysis capability. It leverages the Windows API to collect information directly regarding the volatile state of the endpoints. While not being able to have the same level of visibility as a memory forensic tools that can parse the computer physical memory, it has artifacts that produces similar outputs, but much faster [89]. A full memory capture, which could be used by those memory forensic tools is also possible with Velociraptor.

In this section, the three machine learning algorithms used in this thesis by both the implemented method using Velociraptor and machine learning and the validation method of using Volatility and machine learning were presented. The techniques used to train, fit and test a machine learning model were also covered. This is used in the following section, Methodology, to generate the models. In addition, the theory behind the malware used in this thesis was covered along with their persistence mechanisms. Finally, the tool used in the implemented methodology to collect the artifacts, Velociraptor, was introduced.

## Section 4

# Methodology

This chapter outlines the phases of research required to test the research hypothesis and validate the implemented methodology. The implemented method based on Velociraptor and machine learning is compared to using Volatility and machine learning. In this thesis, those two methodologies are referred to as Velociraptor and Volatility, respectively. It covers the test environment, the software used, the design decisions and a list of the malware used. It is organized following the machine learning pipeline, as presented in Figure 3

### 4.1 Velociraptor and Volatility Methods

In order to validate the aim of the implemented method which will be covered in this section, it needs to be validated with known methodologies. As discussed in Section 2.5, Volatility is the industry standard for memory forensic analysis. Many research such as Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6] used Volatility to extract raw memory data that can be turned into features used to train a machine learning model. The model can then be used to detect the presence of malware, such as ransomware and rootkit. As such, the implemented method, Velociraptor, was compared to Volatility.

Of course, Velociraptor is not a memory analysis framework, while Volatility is. This is a key difference between the two tools. Volatility is limited to analyzing what is available within the memory, while Velociraptor is not. Velociraptor has some visibility over the process memory using the Windows API, though its main purpose is to acquire disk-based artifacts. The reason why Volatility was used as the tool for the validation methodology is that memory forensics research which used machine learning have the most similar research approach to the implemented method.

The implemented method was validated by conducting the same experiment with both Velociraptor and Volatility. The performance of the machine learning models generated using the Volatility features were used as the validation performance comparison metrics; in order to validate the experiment, the implemented method needed to outperform the models trained using features generated with Volatility raw data.

## 4.2 Data Acquisition

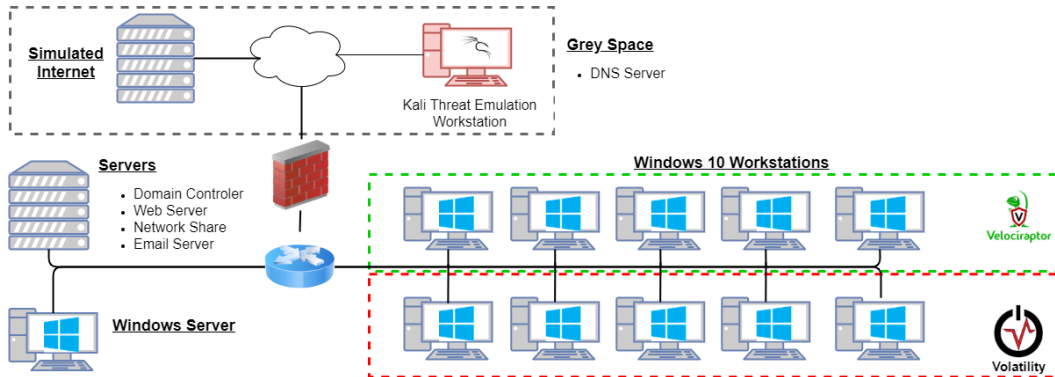
This section covers the steps required to acquire the raw data from the memory and hard disk. This includes creating the network in which the experiment took place, how the experiment was conducted, the different types of malware used and the process of identifying what data had to be collected, or which features were required. Finally, it explains how the data was then collected.

### 4.2.1 Test Environment

In order to compare both Volatility and Velociraptor, a test environment that can simulate an enterprise network was needed. To do so, the Collaborative Security Test Environment (CSTE) from the Directorate Information Management Engineering and Integration (DIMEI) was used. CSTE enabled the creation of a virtual range which has lots of computing resources and uses VMware vSphere.

The network simulates a small company, Globotech, which has ten employees. It contains the basic infrastructure that would be required for such a company to operate. Users are simulated using the Human Actor Like Orchestration (HALO) software, developed by Field Effect Software [90]. This tool was used to add some realism to the test environment and generate simulated background activity would be present within the environment. Details on the use of HALO in this research is covered in Section 4.2.1.2. Figure 33 shows a network diagram of the environment used. Each component of this network, such as the workstations, the servers and grey space are explained in detail in the following sections.





**Figure 33—Test Environment**

#### 4.2.1.1 Workstations

Ten Windows 10 workstations using Version 1909 (build 18363.778) has been set up. To compare both Velociraptor and Volatility, the workstations were separated into two groups of five, for each method. Table 1 shows the breakdown of the ten workstations, with the workstation's name, the role in the company, the username, the level of privilege of the user and the tool used. Since workstations 11 to 15 all had user-level privileges, some malware requiring more privileges needed to be run on a different workstation.

**Table 1—User workstations in the test environment**

<b>Workstation</b>	<b>Role</b>	<b>Name</b>	<b>Account Role</b>	<b>Tool</b>
<b>R9-UA-11</b>	CEO	Alice	Domain User	Volatility
<b>R9-UA-12</b>	Marketing	Bob	Domain User	Volatility
<b>R9-UA-13</b>	Sales	Charlie	Domain User	Volatility
<b>R9-UA-14</b>	HR	Dan	Domain User	Volatility
<b>R9-UA-15</b>	Finance	Frank	Domain User	Volatility
<b>R9-UA-16</b>	IT	Grace	Domain Admin	Velociraptor
<b>R9-UA-17</b>	IT	Henry	Administrator	Velociraptor
<b>R9-UA-18</b>	R&D	Iris	Domain User	Velociraptor
<b>R9-UA-19</b>	R&D	Juliette	Administrator	Velociraptor
<b>R9-UA-20</b>	Logistics	Kyle	Domain User	Velociraptor

4.2.1.2 *Human Actor Like Orchestration (HALO)*

HALO simulates user activity to make the enterprise network more realistic. This includes simulating programs being open, email being received, open and sent, web browsing, etc.

HALO enables the creation of a schedule that can be used to automate user activity. Table 2 shows the schedule used for the Volatility Workstations and Table 3 for the Velociraptor workstations. This was designed so that the host activity would be more varied and that the start and end time of the workday would vary slightly between workstations. This also added more realism to the simulated environment.

**Table 2—Volatility workstations schedules**

<b>Time</b>	<b>Alice</b>	<b>Bob</b>	<b>Charlie</b>	<b>Dan</b>	<b>Frank</b>
7:30	Email			Random Task	
8:00	Office	Email	Email	Web Browsing	Random Task
8:30	PowerPoint	Word	Web Browsing	Word	Email
9:00	Email	PowerPoint	Excel	Email	Excel
9:30	Word	Web Browsing	Email	Email	Excel
10:00	Office	Office	Office	Random Task	Excel
10:30	Office	Email	Office	Email	Word
11:00	Email	Word	PowerPoint	Office	Office
11:30	Web Browsing	Random Task	Excel	Random Task	Office
12:00	Word	Web Browsing	Web Browsing	Web Browsing	Web Browsing
12:30	Email	Random Task	Web Browsing	Random Task	Web Browsing
13:00	Office	Email	Email	Email	Excel
13:30	Office	Random Task	Excel	Excel	Office
14:00	Email	Word	PowerPoint	Office	Excel
14:30	Office	Web Browsing	Office	Random Task	Excel
15:00	Random Task	Office	Email	Web Browsing	PowerPoint
15:30	Random Task	Email	Web Browsing	Web Browsing	Random Task
16:00	Random Task		Email		
16:30	Email				

**Table 3—Velociraptor workstations schedules**

<b>Time</b>	<b>Grace</b>	<b>Henry</b>	<b>Iris</b>	<b>Juliette</b>	<b>Kyle</b>
7:30		SysAdminWork	Email		
8:00	Web Browsing	SysAdminWork	Office	Web Browsing	Email
8:30	Random Task	Random Task	Web Browsing	Office	Excel
9:00	Random Task	Web Browsing	Office	Word	Excel
9:30	SysAdminWork	Random Task	Office	PowerPoint	Web Browsing
10:00	SysAdminWork	Random Task	Random Task	Email	Office
10:30	Random Task	Office	Email	Random Task	Office
11:00	Random Task	SysAdminWork	Office	Random Task	Random Task
11:30	SysAdminWork	SysAdminWork	Word	Excel	Email
12:00	Random Task	Web Browsing	Office	Random Task	Web Browsing
12:30	SysAdminWork	Random Task	Email	Web Browsing	Web Browsing
13:00	SysAdminWork	Random Task	Web Browsing	Email	Office
13:30	Random Task	Office	Web Browsing	Random Task	Office
14:00	Random Task	Random Task	Random Task	Office	Excel
14:30	Random Task	Web Browsing	Random Task	Office	Email
15:00	Random Task	Random Task	Office	Office	Random Task
15:30	Random Task	Email	Email	Web Browsing	Random Task
16:00					Email
16:30					

HALO has been configured to run locally on each computer. Using the schedules in Table 2 and Table 3, a profile was generated so that the programmed activity would occur. Each profile details how a particular user is being simulated.

#### 4.2.1.3 *Windows Server*

In order to gather all the collected data from both Velociraptor and Volatility in one place, a Windows Server was set up to serve as a muster point where data was then taken outside of the environment for analysis.

The collection of the artifacts on the workstations was done every 30 minutes using a scheduled task. The scheduled task runs a batch script that then runs either the Velociraptor offline collector executable, or the Dumpit and Volatility executables. It then runs a PowerShell script that sends all collected artifacts onto the Windows server. This is covered in more details in Section 4.2.3.

#### 4.2.1.4 *Servers*

Within the environment, multiple servers were set up and configured so that the simulated users, emulated by HALO, could function properly. This includes a Domain Controller that contains the Globotech DNS server, a file share, a mail server and a web server.

#### 4.2.1.5 *Grey Infrastructure*

In order to simulate the outside world, a grey infrastructure has been set up. This includes a Grey DNS server that acts as the authoritative DNS server for the environment, as well as a simulated Internet, provided by DIMEI and called Mr. Internet.

Mr. Internet contains hundreds of scraped websites, which increases the fidelity of the environment. The HALO agents can browse the simulated web and get real webpages back and make the user activity look more authentic.

#### 4.2.1.6 Threat Emulation

In order to run the malware, a red infrastructure needed to be set up. Kali Linux was used as the attack platforms. All malware samples were staged from this workstation and the C2 led to it. For the experiment, no redirectors were used as the focus was on host-based activity and the change of IP address to different servers would not have impacted the experiment.

#### 4.2.2 Malware

For the experiment, 11 unique malware binaries and 3 frameworks were used to create 41 different instances or sets of malware behaviour. Table 4 shows the list of malware binaries and frameworks used. The following section covers those programs and the intended effect on the infected host.

**Table 4—Malware executables and tools used**

Malware Name	Malware Type	Malware Behaviour	Number of samples
<b>CatfishHTTPSExfiltrator</b>	Binary	Data Exfiltration	1
<b>Lyonfish</b>	Binary	Ransomware	1
<b>CatfishFileShredder</b>	Binary	RAT	1
<b>CatfishSocket1</b>	Binary	RAT	1
<b>CatfishExplorer</b>	Binary	RAT	1
<b>CatfishPowerShell1</b>	Binary	RAT	1
<b>Living Off The Land</b>	Binary	Persistence	1
<b>CatfishPersister</b>	Binary	Credentials Stealing/Persistence	2
<b>OffensivePH</b>	Binary	Post-exploitation tool	3
<b>77rootkit</b>	Binary	Rootkit	1
<b>Hidden</b>	Binary	Rootkit	5
<b>Metasploit</b>	Framework	RAT	3
<b>PowerShell Empire</b>	Framework	RAT	1
<b>Cobalt Strike</b>	Framework	RAT/Persistence/Credentials Stealing	19

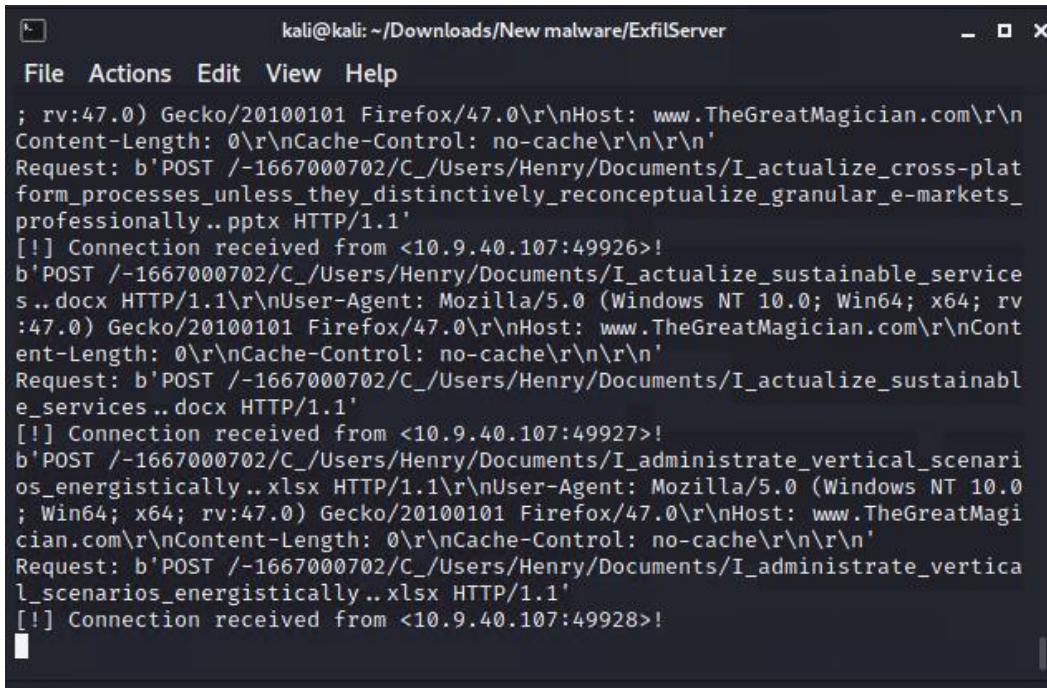
For this research, the detection focus was on the last four steps of the cyber kill chain [13] covered in Section 2.3.1. The malware generated artifacts related to the exploitation, installation, C2 and actions on objective phases of the kill chain as the visibility of the implemented methodology is on the host and there is limited network visibility. As such, it was not expected that the delivery stage would be detectable.

#### 4.2.2.1 *CatfishHTTPSExfiltrator*

CatfishHTTPSExfiltrator is a data exfiltration tool that sends all the files in the users' Documents folder to the remote server.

On the server, a python script is running to listen for incoming connections to the domain "www.TheGreatMagician.com", which resolves to the attacker's server. Once the program is running on the user's workstation, it sends the files over an HTTP/HTTPS tunnel. Figure 34 shows the files being received on the server.

This creates unusual network activity and a surge of files been accessed/modified.











```
kali@kali: ~/Downloads/New malware/ExfilServer
File Actions Edit View Help
; rv:47.0) Gecko/20100101 Firefox/47.0\r\nHost: www.TheGreatMagician.com\r\n
Content-Length: 0\r\nCache-Control: no-cache\r\n\r\n'
Request: b'POST /-1667000702/C_/Users/Henry/Documents/I_actualize_cross-plat
form_processes_unless_they_distinctively_reconceptualize_granular_e-markets_
professionally..pptx HTTP/1.1'
[!] Connection received from <10.9.40.107:49926>!
b'POST /-1667000702/C_/Users/Henry/Documents/I_actualize_sustainable_service
s..docx HTTP/1.1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv
:47.0) Gecko/20100101 Firefox/47.0\r\nHost: www.TheGreatMagician.com\r\nCont
ent-Length: 0\r\nCache-Control: no-cache\r\n\r\n'
Request: b'POST /-1667000702/C_/Users/Henry/Documents/I_actualize_sustainabl
e_services..docx HTTP/1.1'
[!] Connection received from <10.9.40.107:49927>!
b'POST /-1667000702/C_/Users/Henry/Documents/I_administrate_vertical_scenari
os_energistically..xlsx HTTP/1.1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0
; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0\r\nHost: www.TheGreatMagi
cian.com\r\nContent-Length: 0\r\nCache-Control: no-cache\r\n\r\n'
Request: b'POST /-1667000702/C_/Users/Henry/Documents/I_administrate_vertica
l_scenarios_energistically..xlsx HTTP/1.1'
[!] Connection received from <10.9.40.107:49928>!
```

Figure 34—CatfishHTTPSExfiltrator server receiving files on the server

#### 4.2.2.2 *Lionfish*

Lionfish is a ransomware that encrypts the files in the user document folder. Once the encryption is completed, each encrypted file now has the “.evil” extension, as well as a “.evil.txt” file, as shown in Figure 35. Those text files contain the ransom note with the contact information to get the files decrypted.

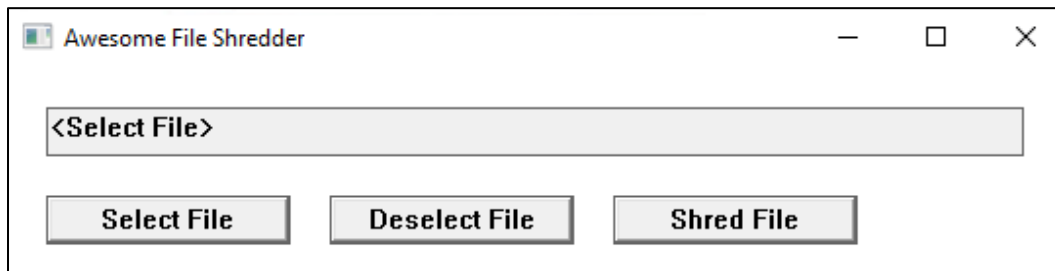
This malware creates a lot of file activity, as all files in the Document folder get encrypted and new files are generated.

	~\$alice@globotech.com.xlsx.evil.txt	2/27/2022 6:15 PM	Text Document	1 KB
	~\$liette@globotech.com.docx.evil.txt	2/27/2022 6:15 PM	Text Document	1 KB
	20.pptx.evil	2/27/2022 6:14 PM	EVIL File	31 KB
	20.pptx.evil.txt	2/27/2022 6:14 PM	Text Document	1 KB
	20.xlsx.evil	2/27/2022 6:14 PM	EVIL File	8 KB
	20.xlsx.evil.txt	2/27/2022 6:14 PM	Text Document	1 KB
	aabuSGtZAC.xlsx.evil	2/27/2022 6:14 PM	EVIL File	8 KB
	aabuSGtZAC.xlsx.evil.txt	2/27/2022 6:14 PM	Text Document	1 KB

**Figure 35—Files on the victim computer after Lionfish encrypted the files in the user’s Document folder**

#### 4.2.2.3 *CatfishFileShredder*

CatfishFileShredder is a trojan that launches a seemingly benign program which deletes the selected file. Figure 36 shows the displayed interface once the program is run on the victim workstation.



**Figure 36—CatfishFileShredder trojan window**

The real behaviour of the program, however, is to create a remote PowerShell session on the victim's workstation. On the Kali workstation, the remote shell can be received with the command "nc -lvp 6060". The command prompt can then be interacted with. In order to generate some activity on the host, the attacker navigated to the user's Documents folder, listed the files, deleted all files in that folder and launched notepad.

This malware creates a lot of file activity, as all files in the Document folder get deleted, in addition to the network connection and the remote session.

#### 4.2.2.4 *CatfishSocket1*

Similar to CatfishFileShredder, this program runs in the background and creates a remote PowerShell session that can be received on the Kali workstation using the command "nc -lvp 5478". The command prompt can then be interacted with. In order to generate some activity on the host, the attacker navigated to the user's Documents folder, listed the files, deleted all files in the folder and launched the notepad application.

This malware creates a lot of file activity, as all files in the Document folder get deleted, in addition to the network connection and the remote PowerShell session.

#### 4.2.2.5 *CatfishExplorer*

CatfishExplorer is a C2 tool that queries the server located at the address "http://FunnyBearJokes.org" every five seconds to know what next task needs to be performed. The tasks are sent using HTML comments in the requested page. For the purpose of the experiment, the malware functionality was used to launch a new program, notepad. The malware decodes the instructions as the command in the comment needs to be base64 encoded.

This led to having multiple instances of notepad being launched at regular intervals, as well as the network connection used for C2. Notepad was not part of the programs HALO would use part of the baseline. In any case, having a new instance of notepad launch every five seconds creates a lot of program activity which can be classified as abnormal.



#### 4.2.2.6 *CatfishPowerShell1*

CatfishPowerShell1 is a malware that downloads a file called “reddit\_redirect.html” off the website “silentreddragon.cn” to the location “C:\ProgramData\reddit.com”. This file could be anything; however, it was decided to use a Meterpreter reverse TCP beacon [71] and name it “reddit\_redirect.html”. This file was then used to connect to the Kali workstation’s Metasploit server, discussed in Section 4.2.2.7.

This file being downloaded to an uncommon location generates artifacts.

#### 4.2.2.7 *Metasploit*

Metasploit is one of the most common penetration testing frameworks. It enables exploitation and post-exploitation of an adversary’s workstations [69].

Three different collections were generated. First, the meterpreter session was established using the beacon “reddit\_redirect.html” downloaded previously from CatfishPowerShell1. The beacon calls back to the Kali server. From there, the following commands were executed: “ls”, “ifconfig”, “execute -f cmd.exe -i -h”, “clearev”, download a file, cat a file. Second, from this initial access, the second collection was done after the beacon migrated to a different process. This is shown in Figure 37. Finally, additional actions on were performed, such as dumping the hash and taking a screenshot of the user workstation.

This created some network activities, as well as the injection of the payload into the new process.

```
meterpreter > run post/windows/manage/migrate
[!] SESSION may not be compatible with this module (missing Meterpreter features: stdapi_sys_process_set_term_si
[*] Running module against R9-UA-11
[*] Current server process: reddit.com (5396)
[*] Spawning notepad.exe process to migrate into
[*] Spoofing PPID 0
[*] Migrating into 408
[*] Successfully migrated into process 408
```

**Figure 37—Process migration using Meterpreter**

#### 4.2.2.8 *PowerShell Empire (PSE)*

PowerShell Empire (PSE) is another post-exploitation framework. While it is no longer actively maintained, it is still widely used. Starkiller [91], the GUI front end can be used to manage the beacons.

Using PSE, a macro embedded office document, “Resume.hta”, was generated that would be used to beacon out to the Kali Linux workstation. An HTA file is an HTML application file that can contain VBScript code [35]. From there, the file was downloaded and executed. The user clicked the “enable macro” pop-up. Then the beacon called back to the attacker’s computer. From the Kali workstation, the beacon appeared in Starkiller. The session was then interacted with by running the “whoami” command and using the file explorer functionality to see the folder structure of the target workstation.

The ran macro and network activity generate artifacts.

#### 4.2.2.9 *Cobalt Strike*

Cobalt Strike is a proprietary adversary simulation and red team operations tool [92]. It is similar to Metasploit and PSE, but has multiple unique scripts and toolsets. Figure 38 shows the GUI for the tool. It can display a visual tree representation of the victim hosts, as well as the individual beacons prompt, where commands can be sent.

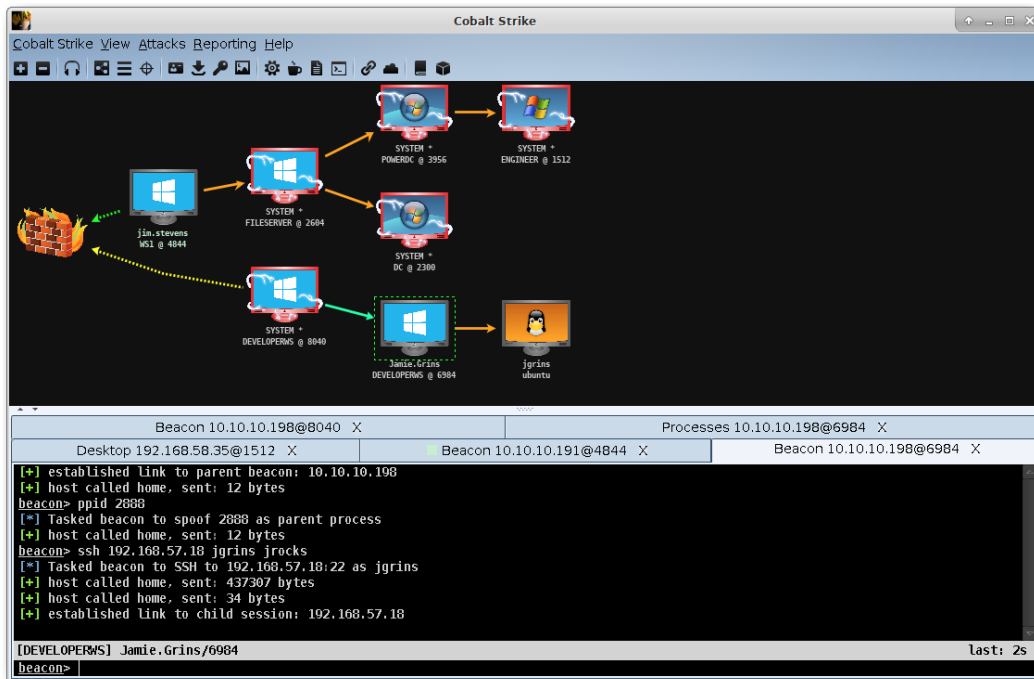


Figure 38—Cobalt Strike main interface (Reproduced from [93])

19 individual collections were generated using Cobalt Strike. This ranged from simply running the beacon to using the multiple functionalities of the tool, such as injecting into a process, keylogging, taking a screenshot, privilege escalation, lateral movement, mimikatz [94], macro-embedded Microsoft Word documents and establishing persistence. Scheduled task, Registry, UserInitMprLogonScript and a service were used as persistence mechanisms during the experiment.

This generated a lot of activity, including some network traffic, process injection, hashes dump and hooks to other processes that should be detected.

#### 4.2.2.10 Living Off The Land

Living Off The Land (LOTL) is a fileless attack platform with persistence developed by Martin Fisher and made available on GitHub [95]. The default binary provided by the author was used as it was sufficient to generate the desired activity on the

victim's computer. This program leverages the Windows LOLBins, such as PowerShell to operate, as covered in Section 3.2.4.

LOTL runs in multiple stages. First it installs itself by writing `Injector.exe` to the registry, then writes an inline PowerShell script to the "HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run" registry, as described in Section 3.3. PowerShell is then run to execute the rest of the malware. The second stage injects into a process. PowerShell loads `injector.exe` and loads the payload executable in a legitimate binary, "svchost.exe", using process hollowing, as covered in Section 3.2.2. As a result, the payload then runs as "svchost.exe" [95].

The process injection as well as the PowerShell activity generates artifacts.

#### 4.2.2.11 *CatfishPersister*

This malware contains two resources that are extracted during the execution. `CatfishPersister` creates "COM542.exe" in "C:\\ProgramData". "COM542.exe" then creates "Krypto.dll" in "C:\\ProgramData". The malware also adds a link to "COM542.exe" to the user startup folder for persistence. Then, the malware operates with two different modes, depending on whether it is run as administrator or as a regular user.

As a user, it accesses the Security Account Manager (SAM) [96] database of the computer to download the password hashes. This is commonly done by attackers as cracking some of the passwords would facilitate lateral movement and privilege escalation.

If run as an administrator, it attempts to download additional malware, "explorer64.exe", from the site "<http://FrozenRedDragon.cn/RedPanda.jpg>", then adds persistence using registry keys.

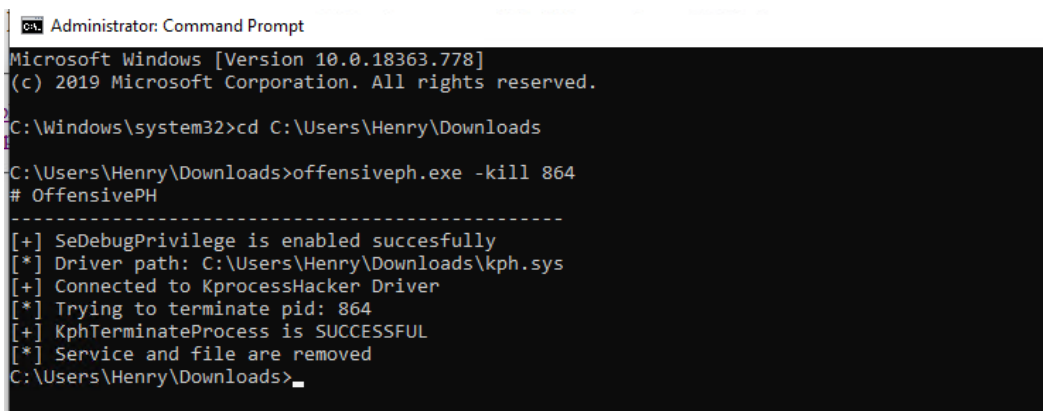
Both modes were used during the collection. The persistence, download of the file and the SAM access generates artifacts.

#### 4.2.2.12 OffensivePH

OffensivePH is another post-exploitation tool and requires Process Hacker [97] to be installed as it leverages its resources. It loads them into the current working directory as kph.sys. As such, during the three collections, Process Hacker had to be first installed on the host. OffensivePH uses its “Hook2Kph.dll”, which can then be injected in any running process [98].

The three collections covered killing a process, injecting shellcode by using hijack thread execution and injecting shellcode into a new “services.exe” instance. Figure 39 shows the functionality that kills a process.

The process injection and the act of killing a process generate artifacts.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\Henry\Downloads

C:\Users\Henry\Downloads>offensiveph.exe -kill 864
# OffensivePH
-----
[+] SeDebugPrivilege is enabled succesfully
[*] Driver path: C:\Users\Henry\Downloads\kph.sys
[+] Connected to KprocessHacker Driver
[*] Trying to terminate pid: 864
[+] KphTerminateProcess is SUCCESSFUL
[*] Service and file are removed
C:\Users\Henry\Downloads>
```

Figure 39—OffensivePH killing a process

#### 4.2.2.13 77rootkit

77rootkit (R77) is a user-mode registry-resident rootkit that can hide files, directories, processes, CPU usage, registry keys and values, services and network connection on the infected host [99].

R77 first stage consists of installing itself by creating two scheduled tasks that run at the system startup. Those tasks run a one-line PowerShell script, which runs the rootkit executable saved in the registry key. Then, it uses process hollowing to create its service

process that has winlogon.exe as the parent process. The code for the R77 service executable is directly loaded.

R77 uses the Detours library, which was developed by Microsoft in order to expand on the OS and application functionalities and make program modifications easier for developers. However, it can also be used by malware authors to attach a malicious DLL to an existing program and create hooks. It is especially useful to hook a rogue DLL to a legitimate program on disk. In Figure 40, the PEview tool shows the Detours section of the executable where the malicious DLL “evil.dll” has been attached [23]. R77 uses exactly this method to avoid writing to disk.

The different hooks created by R77 generate artifacts.

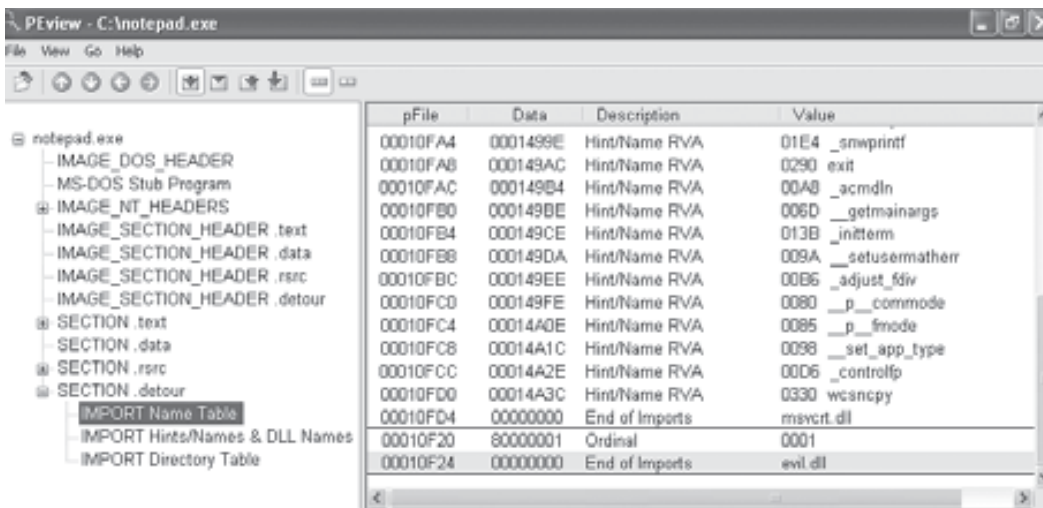


Figure 40—PEview of Detours with a rogue DLL, evil.dll (Reproduced from [23])

#### 4.2.2.14 Hidden

Hidden is another rootkit that can hide files, registry keys and values, directories and processes. It is a rogue driver that has to be installed with the Windows digital signature enforcement disabled. Then, it runs as a service on the computer [100].

The malware comes with a command line utility that can be used to hide and unhide elements. Five collections to test the different features were generated. First, simply

installing the driver. Then, the collections were taken after hiding a file, a directory, a registry key and a process. Figure 41 shows the service running and used to hide the file “calc.exe”.

The rogue driver can be detected as it is unsigned and is getting installed.

```
C:\Windows\system32>sc start hidden

SERVICE_NAME: hidden
        TYPE               : 2  FILE_SYSTEM_DRIVER
        STATE                : 4  RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE      : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT          : 0x0
        WAIT_HINT           : 0x0
        PID                 : 0
        FLAGS                :
C:\Windows\system32>cd C:\Users\Alice\Desktop\hidden\x64\Release
C:\Users\Alice\Desktop\hidden\x64\Release>HiddenCLI.exe /hide file c:\Windows\System32\calc.exe
Command 'hide' successful
status:ok;ruleid:1
```

**Figure 41—Hidden rootkit hiding a file**

#### 4.2.3 Features Extraction

Before the data could be collected, features extracted during the collection needed to be selected as this would determine exactly which plugins would be required during the collection process. The feature extraction process was done for Volatility by replicating recent research experiments as this served as the validation and is explained in the following section, Section 4.2.3.1. For Velociraptor, this was done by following recent research methodology and doing experimental testing to find features that could detect the selected malwares. This is covered in Section 4.2.3.2.

##### 4.2.3.1 Volatility

For Volatility, most of the features from Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6] were used in the validation. From Cohen and Nissim [4], all 23 features were used, as shown in Table 5.

**Table 5—Features from Cohen and Nissim [4] (Reproduced from [4])**

ID	Feature name	Description	Type	Volatility plugin
1	callbacks_num	Number of kernel callbacks.	Integer	callbacks
2	dlls_dllist_unique_paths_num	Number of unique DLLs used by all processes.	Integer	dllist
3	dlls_ldrmodules_num	Number of DLLs used by all processes.	Integer	ldrmodules
4	dlls_ldrmodules_unique_mappedpaths_num	Number of unique DLLs used by all processes.	Integer	ldrmodules
5	dlls_ldrmodules_InInit_false_num	Number of DLLs that were not found in the <i>InInit</i> list of process memory.	Integer	ldrmodules
6	dlls_ldrmodules_InLoad_false_num	Number of DLLs that were not found in the <i>InLoad</i> list of process memory.	Integer	ldrmodules
7	dlls_ldrmodules_InMem_false_num	Number of DLLs that were not found in the <i>InMem</i> list of process memory.	Integer	ldrmodules
8	dlls_ldrmodules_all_false_num	Number of DLLs that were not found in the <i>InInit</i> , <i>InLoad</i> , and <i>InMem</i> lists of process memory.	Integer	ldrmodules
9	modules_num	Number of driver objects associated with kernel modules.	Integer	modules
10	processes_privs_enabled_not_default_num	Number of privileges (of all processes), which have the <i>Enable</i> attribute and do not have the <i>Default</i> attribute.	Integer	privs
11	processes_psxview_exited_num	Number of processes that were previously running but were completed before the snapshot was taken.	Integer	psxview
12	processes_psxview_false_columns_num	Number of process listing techniques that do not detect at least one process.	Integer	psxview
13	processes_psxview_false_rows_num	Number of processes that are not detected by at least one process listing technique.	Integer	psxview
14	processes_psxview_num	Number of processes detected by the <i>psxview</i> plugin.	Integer	psxview
15	processes_psxview_okay_rows_num	Number of processes that are not detected by at least one process listing technique, and are valid exceptions (each process should be detected by all techniques).	Integer	psxview
16	processes_psxview_pslist_true_num	Number of processes detected by the <i>pslist</i> plugin (appear in <i>psxview</i> plugin output).	Integer	psxview
17	processes_psxview_psscscan_true_num	Number of processes detected by the <i>psscscan</i> plugin (appear in <i>psxview</i> plugin output).	Integer	psxview
18	services_svcsan_num	Number of services (running and stopped).	Integer	svcsan
19	services_svcsan_running_num	Number of running services.	Integer	svcsan
20	services_svcsan_stopped_num	Number of stopped services.	Integer	svcsan
21	handles_num	Number of handles.	Integer	handles
22	mutex_mutantscan_num	Number of mutexes in use.	Integer	mutantscan
23	threads_thrdsan_num	Number of threads.	Integer	thrdsan

Wang et al. [5] focused their work on rootkit detection and had slightly different features set. Table 6 shows, out of their features, the ones used in this thesis. Most of these features were reproduced except the features which used the orphan thread, hooks in the I/O Request Packet (IRP) and malicious attachment plugins as they were not collected.

**Table 6—Features from Wang et al. [5] used in this thesis (Reproduced from [5])**

ID	Feature	Description	Data type	Plugins
1	Hidden modules	Whether hidden modules exist	Boolean	modules, modscan
2	Driver object	Whether abnormal driver object exists	Boolean	driverscan, modules
3	SSDT hooking	Whether there are hooks on SSDT	Boolean	ssdt
4	Abnormal Callbacks	Whether there is malicious callback in the system	Boolean	callbacks
5	Abnormal timers	Whether there is malicious timer in the system	Boolean	timers



Panker and Nissim [6] used 177 features to detect unknown malware on Linux cloud environments. Some features were adapted to monitor a Windows system. Table 7 shows the list of the twelve features used based on their research. Those twelve features were all generated from the pslist and netstat plugins. Since [6] focused on Linux malware, most of the features could not be adapted.

**Table 7—Features added based on Parker and Nissim [6]**

<b>ID</b>	<b>Feature</b>	<b>Description</b>	<b>Data Type</b>	<b>Plugin</b>
1	net_conn_amount	number of active network connections	int	netstat
2	tcp_conn_amount	number of active TCP	int	netstat
3	udp_conn_amount	number of active UDP	int	netstat
4	listen_amount	number of listen connections	int	netstat
5	established_amount	number of established connections	int	netstat
6	close_wait_amount	number of close wait status connections	int	netstat
7	close_amount	number of close status connections	int	netstat
8	different_tcp_port	number of different TCP ports	int	netstat
9	different_udp_port	number of different UDP ports	int	netstat
10	empty_process_name	number of processes es with empty names	int	pslist
11	child_PowerShell	number of processes named 'PowerShell'	int	pslist
12	child_cmd	number of processes named 'cmd'	int	pslist

The amalgamation of the features from the three research increases the level of visibility over the hosts as each research focused on different types of malware. During the Exploratory Data Analysis in Section 4.3.1, the effectiveness of each of those features to predict the tested malware is measured.

#### 4.2.3.2 *Velociraptor*

The Velociraptor feature extraction process was done by looking at similar research using disk-based forensic and memory forensic. This section covers the initial set of features used.

In total, 76 plugins were extracted by looking at what was done for Volatility and in disk forensic-based research. In addition, experimental tests were performed using the Velociraptor GUI, which was run in a separate test environment for testing purposes, was used on a host where the malware was executed to see which plugins would increase visibility over a particular type of malware.

For memory forensic inspired plugins, a methodology similar to Cohen and Nissim [4] and Panker and Nissim [6] was used to identify the features that would be useful.

From Cohen and Nissim [4], multiple DLL, handles and services related features which extract information similar as shown in Table 5 were selected. However, the lack of a psxview plugin in Velociraptor made multiple features impossible to collect as no alternatives exist in Velociraptor. From Panker and Nissim [6], multiple network-related features were selected, in addition to some processes related plugins.

For disk-based forensic, Mohammad and Alqahtani's [12] work, which looked at machine learning techniques for file system forensic analysis, was used to select file-based plugins. Table 8 shows the features they used to perform a disk-based forensic analysis with machine learning.

**Table 8—Features considered by Mohammad and Alqahtani [29] (Reproduced from [12])**

No	Description	Data type
1	Event date	Date
2	Event time	Time
3	User	Text
4	Computer name	Text
5	Event ID	Numerical
6	Type	Text
7	C Time – File creation	Time
8	A Time – File altered	Time
9	M Time – MFT changed	Time
10	R Time – File read	Time
11	DOS File Permissions ( <i>Read-only, Hidden, System, Archive, Temporary, Compressed, Offline, Encrypted</i> )	Text
12	Owner Id	Numerical
13	Allocated size of the file	Numerical
14	Real size of the file	Numerical
15	Flags ( <i>Directory, Compressed, Hidden</i> )	Text
16	Filename length in characters	Numerical
17	Filename namespace ( <i>POSIX, Win32, DOS, Win32 &amp; DOS</i> )	Text
18	Object Id ( <i>Unique Id assigned to file</i> )	Numerical
19	Birth volume Id ( <i>Volume where file was created</i> )	Numerical
20	Birth object Id ( <i>Original object Id of file</i> )	Numerical
21	Domain Id ( <i>Domain in which object was created</i> )	Numerical
22	Access control type ( <i>Access allowed, Access denied, System audit</i> )	Text

Finally, additional plugins unique to Velociraptor were selected if they increased visibility over the host to detect the malware used. Those were determined based on the experimental tests, where malware was run on a host with Velociraptor running. Analyzing the malware behaviour with the Velociraptor GUI manually, such as the API hooking, it was possible to select additional relevant features which would assist the detection. Using the Velociraptor GUI, it is possible to create and edit artifacts, as well as view the details related to the artifacts. Velociraptor has many artifacts that can gather information on some Windows events, registries and the processes. All features from the papers covered in this section helped to select which plugins would be required to generate the features; in total, 40 plugins were deemed required. The complete list of those plugins and features is listed in Annex A.

Following Murthaja et al. [2] methodology of categorizing features, Velociraptor plugins were selected to achieve a similar level of visibility on the host, by looking at Registries, DLLs, API and Network-related artifacts. Table 9 shows the features generated by Murthaja et al. [2]. Velociraptor visibility over the API domain is very limited. As such, the hollows\_hunter [101] tool, available on GitHub, was used. This tool can be run with

Velociraptor using a custom plugin, Custom.Windows.Detection.ProcessHollowing [102], and scans the running processes to look for hooks and in-memory patches. This tool makes it easier to detect process injection and process hollowing.

**Table 9—Features considered by Murthaja et al. [2] (Reproduced from [2])**

<b>Domain</b>	<b>Features</b>
Registry	Registry handles, open handles details, shimcache, pslist handles, pslist threads
DLLs	DLL load time, DLL path, DLL base, ldr modules, ldr mapped path, ldr base, envvars process, envvars base, privs process, getsids process
APIs	APIhooks module, APIhooks functions, APIhooks data, threads details, assembly flags, malfind flags
Network	Slack data, destination IP, destination port, source port, flags

In addition to the four domains shown in Table 9, two other domains were added: a file system category, which includes all the files-based features generated similarly to Mohammad and Alqahtani [12] features in Table 8, and a Windows event category, which includes PowerShell events, remote login events, etc. A full breakdown of the 76 features is available in Table A-. All selected Velociraptor features fall into one of those six domains, as shown in Table 10. In addition, the number of features per domain is identified.

**Table 10—Type of features generated for Velociraptor**

<b>Domain</b>	<b>Features</b>
<b>Registry</b>	24
<b>DLL</b>	7
<b>API</b>	1
<b>Network</b>	13
<b>File System</b>	18
<b>Events</b>	13

#### 4.2.4 Data Collection

In order to generate sufficient data for both experiments, more than a week’s worth of data was needed. This section covers the methodology used to collect the data.

##### 4.2.4.1 *Volatility*

For Volatility, the first step was to capture the memory of the host. Dumpit [39], which creates a physical memory capture that can then be used by Volatility for the analysis, was used. Volatility was run directly on the host under observation in order to reduce the bandwidth requirement. As such, the memory capture remained on the host and the Volatility plugins were run. Their output was then sent to the Windows server and the capture was deleted. A Volatility portable executable that could be run on Windows using the method in [103] was used to analyze the capture. Using Dumpit and Volatility on the host created artifacts. However, since it was common to the baseline and the malware collection, it mitigates the impact of this. This is an issue with live memory forensics collection [4]. However, this significantly reduce the bandwidth requirement, which was one of the objectives of this research.

To collect the Volatility plugins, the artifacts used by Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6] were used, as discussed in Section 4.2.3.1.

A Windows scheduled task was set up on all five Volatility workstations. This scheduled task was set to run every 30 minutes. However, all five Volatility workstations collections were offset by five minutes. This was done to make sure the virtual environment

would be able to support Dumpit. Having multiple memory captures done simultaneously could have taxed the infrastructure as in reality, the same hardware is used to host all the workstations. Multiple simultaneous read and write operations on the disk could have created a bottleneck and slowed the collection process. This scheduled task launched a batch file as administrator, running Dumpit, then a PowerShell script that collected the wanted plugins. The output from the various plugins was saved in a folder named with the hostname, time and date of the collection as JSON files. Finally, the script sent the compressed folder containing the JSON of this collection to the Windows server.

For the malware samples, this scheduled task was disabled and the collection was launched manually after the malware had started or completed its execution. This slightly modified batch file sent the folder with the JSON files to a different repository on the same server, specific for the malware collections.

#### 4.2.4.2 *Velociraptor*

The first challenge for Velociraptor is that the tool is designed to be operated in a client-server architecture. The goal is to be able to query all the clients as if they were a database using VQL. In addition, hunts can be scheduled to collect the host data at a specific time. In order to follow the machine learning pipeline and get the data outside of the environment, a different configuration was needed.

##### 4.2.4.2.1 Offline Collector

The Velociraptor offline collector was used instead of the Velociraptor Server because it was easier to gather all the data in the test environment and then process it with machine learning outside of it. This way, the data was collected directly on the host and sent as CSVs and JSON files to a central location. Those were then later parsed. Using the offline collector also forces the collection to be done now vs. scheduling a hunt, which can take longer to launch based on availability of the host resources. Velociraptor hunt are not necessarily run at the time requested, compared to Windows scheduled task, as the Server communicates with the agent of the hosts to see if the collection can be done now. This can add delay and makes collecting at 30 minutes intervals more difficult.

This offline collector can be generated from the Velociraptor Server. It packages all the requested plugins, even custom plugins generated by the analyst, and generates a single executable that creates a ZIP file with all the requested collection when run as administrator on the target workstation [104].

#### 4.2.4.2.2 Methodology

To collect the Velociraptor artifacts, a Windows scheduled task was set up on all five Velociraptor workstations. This scheduled task was set to be running every 30 minutes, on the hour and on the half hour. This scheduled task launched a batch file as administrator, running the offline collector. Then, once the ZIP file was completed, a PowerShell script was run to move the compressed folder to the Windows Server.

For the malware sample, this schedule task was disabled and the collection was launched manually after the malware had started or completed its execution. This slightly modified batch file sends the collected ZIP to a different repository, specific for the malware collections.

#### 4.2.5 Experimentation

This section discusses how the overall experiment was set up and covers specifically how the data was collected. The details on the malware used for the experiment is covered in Section 4.2.2 and the selected features and plugins used is covered in Section 4.2.3.

The first step was to collect a baseline of the network, by collecting a regular user's activity. The HALO users followed their schedule, which included opening, creating documents, receiving and sending email, etc. In the second step, the malware is executed to generate the malicious artifacts. The third step was to split the data into a testing and a training set. Then, the fourth step was to process the data by the machine learning algorithms. This includes the feature selection, hyperparameter tuning, model training and model evaluation steps of the machine learning pipeline shown in Figure 3. Finally, the

fifth step was the classification of the results of the test set, which were generated and analyzed. The experiment flow is shown in Figure 42.

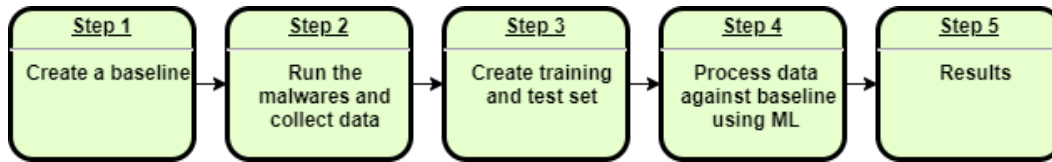


Figure 42—Experiment Flow

The following two sections cover the baseline collection and malware collections.

#### 4.2.5.1 *Baseline Collection*

The baseline was generated by running the Velociraptor Offline Collector, discussed in Section 4.2.4.2.1, and the Volatility collector, discussed in Section 4.2.4.1. In order to account for the time required to run the collector, a 30 min interval between collections was selected. With this interval, for five computers, 840 samples per week were created. Each sample contains all plugins, from either Velociraptor or Volatility, that were collected from the computer at a specific time. Using those plugins, the features were generated; each sample is therefore linked to a data point that can be used for the test set or the training set. In total, 1340 samples for Velociraptor and 2831 samples for Volatility were collected to create the baseline. More samples were collected for Volatility as the collection setup was completed earlier and some technical issues were encountered initially with the Velociraptor collection; the offline collector created some temporary files in Windows and this filled up the hard drive and crashed the host after a few days of collection. All data collected up to the crash was still valid, but the Velociraptor workstation had to be reverted every three days to avoid this issue. In the future, a solution to this problem would be to add in the collection script code that removes the previously generated temporary files. For this reason, since the collection of Volatility was still ongoing while the Velociraptor collection was troubleshooted, more data was available for



the validation experiment using Volatility. Instead of deleting samples, it was decided to keep all data samples.

#### 4.2.5.2 *Malware Collections*

Once the baseline was collected, 41 malware samples, covered in Section 4.2.2, were collected. Each malware sample is a unique use of a malware instance. For example, using Cobalt Strike, 19 unique malicious activities were performed and captured as individual malware sample. To do this, R9-UA-11 and R9-UA-17 workstations were used. In each case, the malware executable was downloaded from the rogue server from the hosted HTTP server on the Kali Linux workstation, then executed as a regular user or as an administrator, based on the desired behaviour. The collection scripts were run and after each occurrence, the computers were reverted to a clean state; for the malware collection, the collections were not run routinely every 30 minutes for the same period as the baseline collection, but manually after the malware had executed.

Once this data was collected on the Windows 10 server in the CSTE environment, the data was taken outside of the environment so it could be analyzed. This is explained in detail the following section, Section 4.2.3.

### 4.3 Exploratory Data Analysis and Data Munging

The Exploratory Data Analysis (EDA) phase focused on looking at the data and identify trends. This section covers how this was done for both Volatility and Velociraptor.

#### 4.3.1 Data Munging

The data collected using Velociraptor and Volatility was normalized using a min-max scalar as this helps improve the accuracy of machine learning models, as covered in Section 3.1.4. Boolean data was transformed into “1” and “0”.

### 4.3.2 Volatility

A total of 40 features were collected for Volatility. Using the built-in SKLearn python class *feature\_importances*, of tree-based classifiers, covered in Section 3.1.3, the most useful features were identified by plotting a graph of feature importance for better visualization. Figure 43 shows the top 20 features using the *feature\_importances\_* values in terms of relevance for Volatility. Process created by cmd.exe and PowerShell were the most useful, followed by the number of loaded modules, some of psxview columns and the number of handles. This was used to train and test the model with a reduced data set to attempt to increase performance. Superfluous features can have a negative impact on the model performance as described in Section 3.1.5.1.

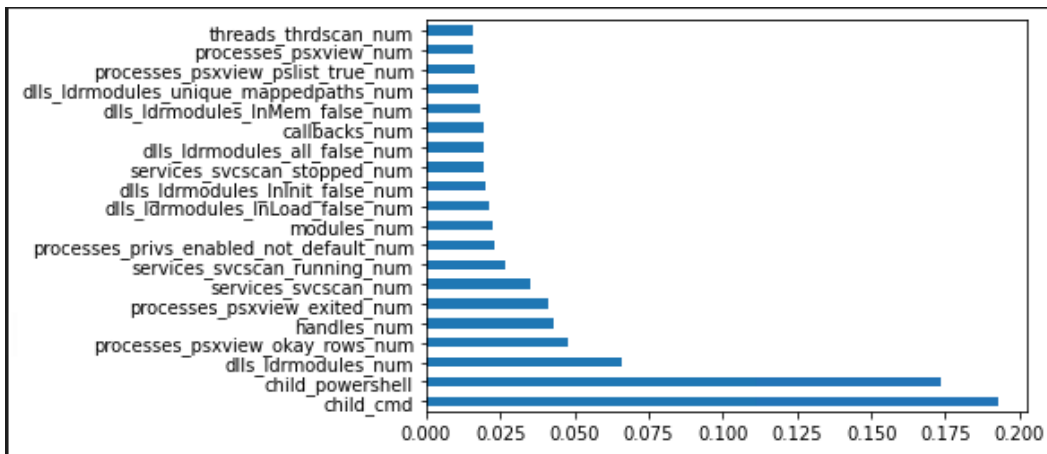


Figure 43—Volatility features importance, using SKLearn *feature\_importances* class

However, looking at the correlation matrix in Figure 44 and the PPS matrix in Figure 45 for Volatility, a lot of the features did not correlate. For the correlation matrix, shown at the top, there are some features that heavily correlate with each other, such as features generated from ldr\_modules, psxview and svcscan. This means that some of those are redundant and not all required. In addition, the row containing the label shows which feature correlates to it, which means which features can best predict the label. The top two values *child\_cmd* and *child\_powershell* both have a value of 0.25, which shows poor

correlation. Multiple implementations that use the correlation matrix to select their features use values of at least 0.5 [105], [106]. This suggests a weak correlation within this dataset.

The PPS matrix also shows a relatively low correlation, as most of the values are zero. To select a reduced set of featured, all feature that correlated with the label were selected. In the case of Volatility, this means only `child_cmd`, which looks at the number of processes that have `cmd.exe` as a parent, has a positive PPS score. This means that according to PPS, only this feature would help to classify the data. This is an indication that the set of features from the three papers may not be optimal for the detection of the tested malware.

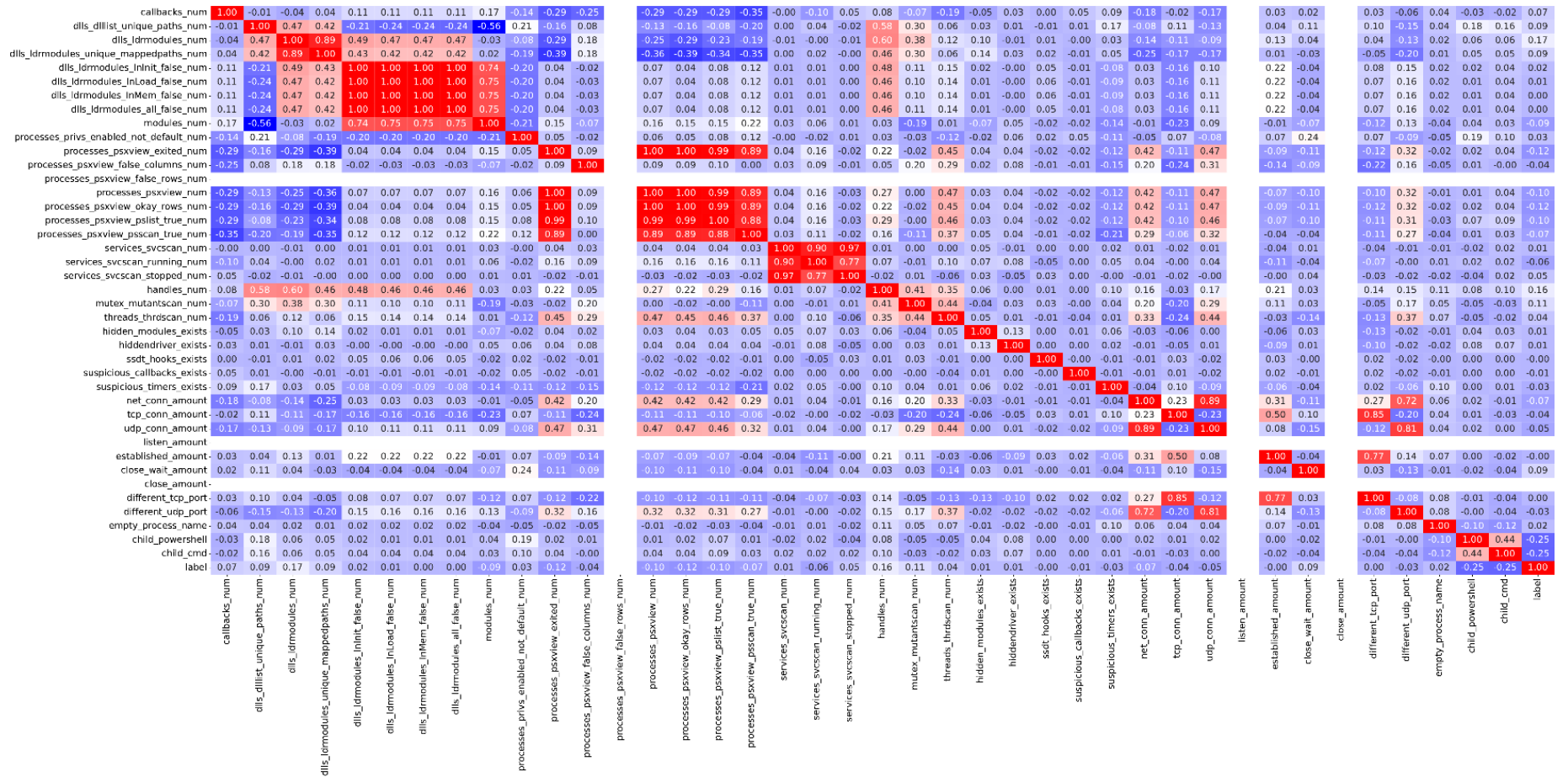


Figure 44—Volatility features Correlation matrix



### 4.3.3 Velociraptor

Out of the 76 features, using the built-in SKLearn python class *feature\_importances* of tree-based classifiers, the most useful features were identified by plotting a graph of feature importance for better visualization. Out of those features, the top 20 most useful features are located shown in Figure 46. The number of untrusted binaries and the LNK files pointing to a remote computer were the two most useful features. As for Volatility, a reduced feature set was used using this technique.

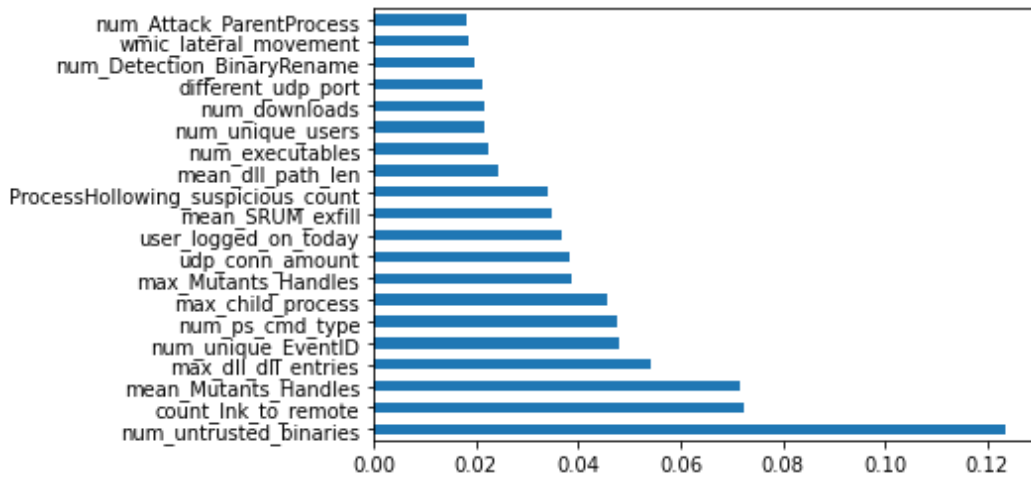


Figure 46—Velociraptor features importance, using SKLearn *feature\_importances* class

Similar to Volatility, the correlation matrix in Figure 47 and the PPS matrix in Figure 48 for Velociraptor shows that a lot of the features do not correlate. For the correlation matrix, there are some features that heavily correlate with each other, such as the named pipes features from the handles plugin and the child process related features from the pstree plugin. The correlation matrix, however, showed a better correlation with the label, with multiple features having a correlation factor above 0.5, such as num\_untrusted\_binaries with 0.69 and mean\_Mutant\_Handle with 0.64. This is better than what was obtained for Volatility and means the features can better classify the data.

For the PPS matrix, while most of the values were still zero, there was more correlation. To select a reduced set of features, all features that correlated with the label

were selected. Out of Figure 48, 16 columns had a value greater than 0 for the row “label”, which mean they showed correlation in predicting the label and would help classify the data. Once the label removed, this led to a reduced feature set of 15 features, listed in Table 11. Again, this is better than observed for Volatility using the same method.

**Table 11—Velociraptor features selected using PPS**

<b>ID</b>	<b>Feature</b>
1	udp_conn_amount
2	num_unique_EventID
3	count_Ink_to_remote
4	num_pipes_names
5	mean_dll_path_len
6	num_ps_cmd_type
7	Mean_Forensics_Bam
8	mean_Mutants_Handles
9	num_Attack_ParentProcess
10	num_Detection_BinaryRename
11	max_child_process
12	max_Mutants_Handles
13	max_URL_entropy
14	max_dll_dll_entries
15	URL_num



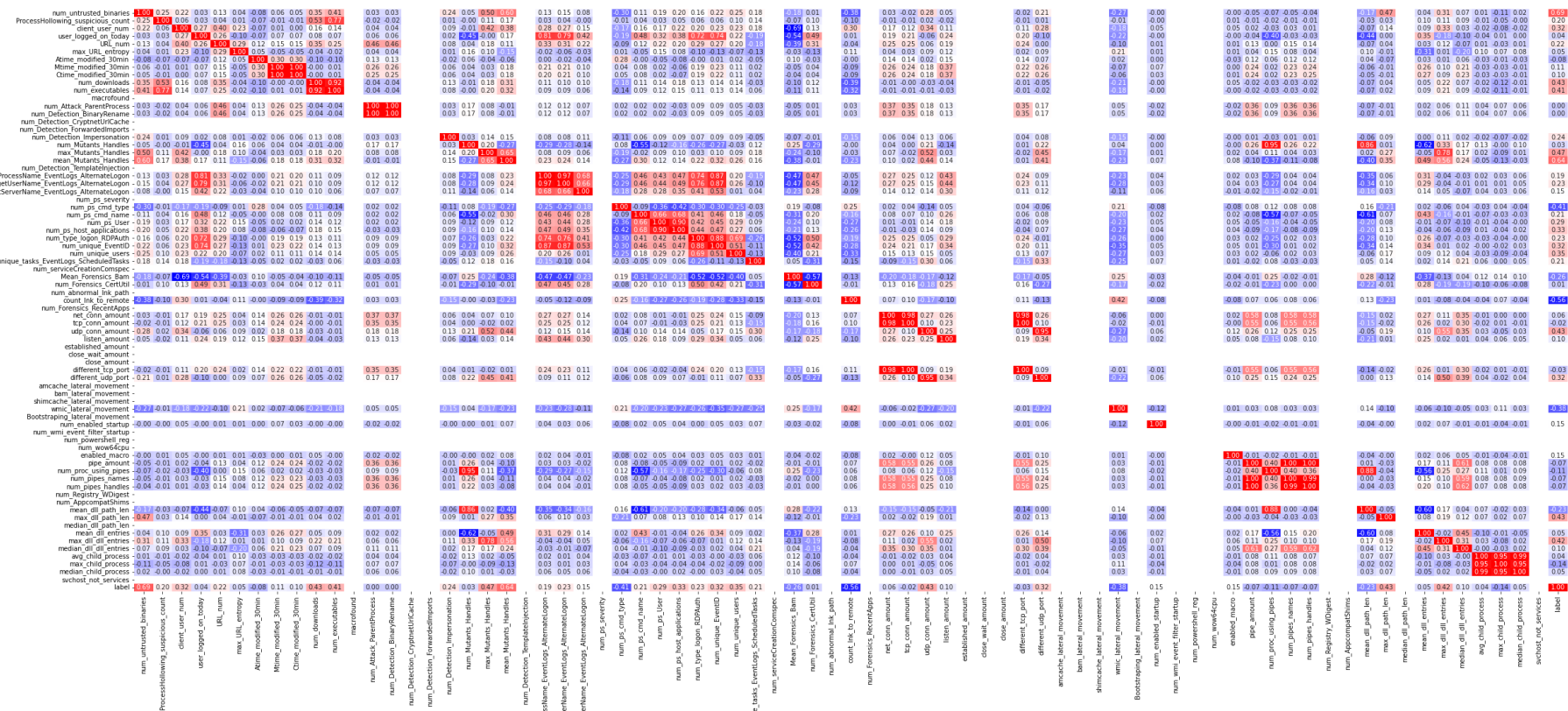


Figure 47—Velociraptor features correlation matrix



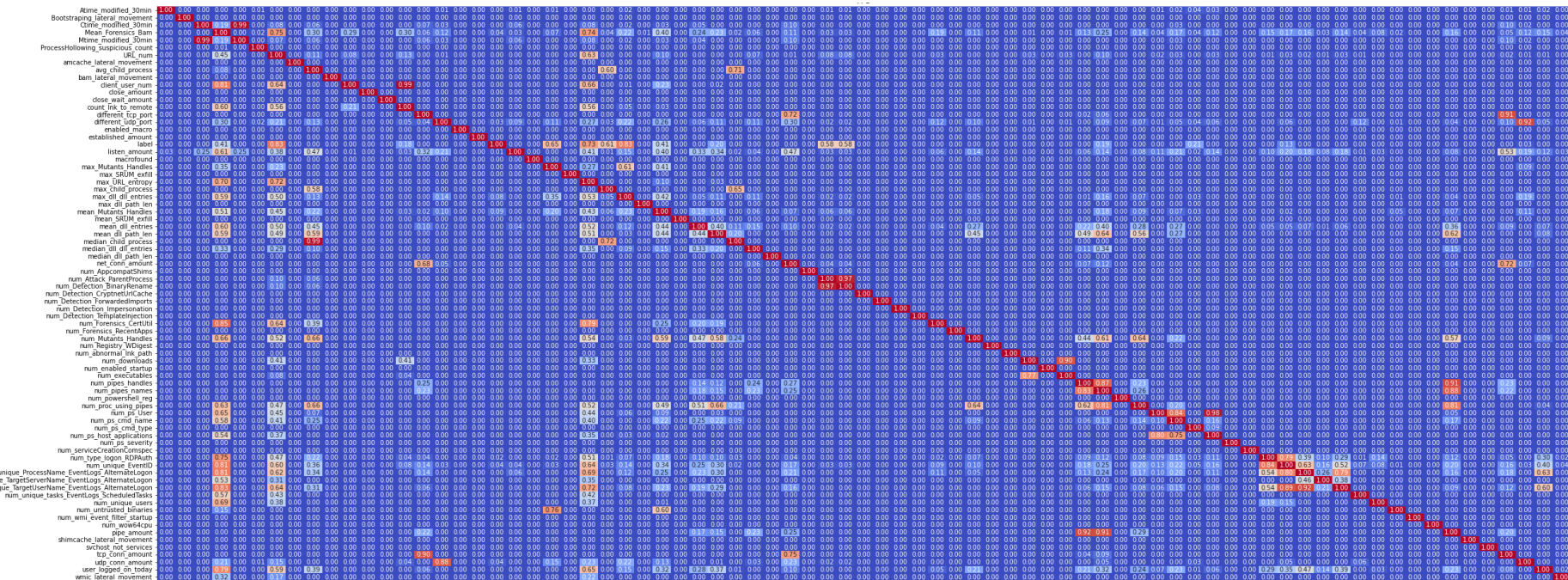


Figure 48—Velociraptor features PPS matrix

## 4.4 Feature Engineering

This section covers the different feature engineering decisions taken to prepare the data for the three machine learning algorithms, including feature reduction in Section 4.4.1 and the cross-validation in Section 4.4.2.

### 4.4.1 Feature Reduction

During the feature engineering phase, five versions of the collected dataset were used to test the performance of the algorithm. Those testing sets were using all the collected features, using the PPS matrix score to keep only the features having a positive correlation with the label, using PCA, as covered in Section 3.1.2.1 to keep 95% and 99% of the variance and selecting the top 20 features using the *feature\_importances* class, shown in Figure 43 for Volatility and Figure 46 for Velociraptor.

Each machine learning algorithm was tested against each of those five datasets to assess performance and the impact of the feature reduction process.

### 4.4.2 Cross-Validation

For the cross-validation, the simpler test-train split referenced in Section 3.1.5.4 was used. The use of a fivefold cross-validation for the dataset tuning would have generated folds with only four malware samples. This would not have been sufficient to properly train the model against the various types of malware used.

The datasets were split in half, one half for the training dataset and the other for the test dataset. This was done due to only having 41 malware collections which left 20 to 21 samples per set. Using a larger training set would have led to fewer malware samples to be tested. Since those malware samples were not all of the same type and generated different type of artifacts, this 50% split was selected. For the same reason, no validation sets were used as it would have required even more data.

This is sufficient for this proof of concept; however, more malware samples of each type would be required to perform a proper cross-validation using k-fold and improve performance.

## 4.5 Model Learning and Evaluation

This section covers the selected machine learning algorithms, the hyperparameters tuning process for each algorithm and how the various datasets were trained.

### 4.5.1 Model Selection

Based on the research, three machine learning algorithms were selected: Isolation Forest, Random Forest and SVM.

Isolation forest has been used in network traffic analysis to find anomalies, such as in Spiekermann and Keller [10]. In this case, Isolation Forest attempts to detect the malware instances as outliers.

Random Forest and SVMs have been used in multiple forensic research. In Mohammad and Alqahtani [12], they compared multiple machine learning algorithms to analyze features collected during a forensic analysis of a file system. From this analysis, they found that RF was the most effective algorithm. However, while SVMs were not as effective, they achieved good performance. Figure 49 shows the results of their research and the effectiveness of some algorithms in this context. Cohen and Nissim [4] also concluded that RF was the most effective algorithm for their application.

Algorithm	Accuracy	Precision	Recall	F1-Measure
NN	89.0297%	88.00%	89.30%	88.70%
SVM-RBF	84.2772%	85.50%	81.70%	83.60%
SVM-PLY	87.4455%	85.90%	88.00%	86.90%
RF	89.0297%	89.30%	88.00%	88.60%
CART	87.8416%	86.30%	88.40%	87.40%
BN-SIM	87.2475%	87.90%	85.30%	86.60%
BN-BMA	87.4455%	87.50%	86.20%	86.80%

**Figure 49—Comparison of the effectiveness of multiple machine learning algorithms for a file system analysis (Reproduced from Mohammad and Alqahtani [12])**

As such, these three algorithms have been selected to classify the generated datasets and classify the data.

#### 4.5.2 Hyperparameter tuning

In order to find the best hyperparameters values, three tuning methods were used: the default parameters, Grid Search and a manual exhaustive implementation. Those three methods are covered in this section.

##### 4.5.2.1 *Default Hyperparameters*

Each algorithm can be used without specifying any hyperparameter values. In that case, the default values are used by the algorithm. The three algorithms were trained with the default values as a reference point. It is not expected to be optimal. As such, it was used only once on the all collected feature set.

##### 4.5.2.2 *Grid Search*

Grid Search was used for each algorithm to find the best hyperparameter values. The most influential parameters were determined based on experimental testing and research.

For Isolation Forest, the tuned hyperparameters were the number of estimators of the model, the maximal number of samples to be used to train estimators, the contamination

rate, which is the percentage of outliers in the model and the number of random states, which affects how the data is split in the different branches [107]. Figure 50 shows the considered values for each of the four parameters.

```
param_grid = {'n_estimators': [10,100,500],
              'max_samples': [10,100,500],
              'contamination': [0.01, 0.02, 0.03, 0.04, 0.05],
              'random_state': [1,10,50]
             }
```

Figure 50—Isolation Forest Grid Search

For Random Forest, more hyperparameters were tuned as more factors could impact the model, and that Random Forest has more hyperparameters that can be tuned. The number of estimators of the model, the maximum number of features the model analyzes before making a decision on where to make a split, the maximum depth of the tree, the criterion on which the effectiveness of a split is evaluated such as impurity (gini) and entropy, the minimal number of samples that each split must contain, the minimal number of samples a node needs to generate a split and the random state [60]. Figure 51 shows the considered values for each of the seven parameters.

```
param_grid = {'n_estimators': [10,100,500],
              'max_features': ['auto', 'sqrt', 'log2'],
              'max_depth': [2,3,4,5,6],
              'criterion': ['gini', 'entropy'],
              'min_samples_split': [4,5,6,7,8],
              'min_samples_leaf': [2,5,10],
              'random_state': [1,10],
             }
```

Figure 51—Random Forest Grid Search

For SVM, three hyperparameters were tuned: the kernel used by the model to create the hyperplane, the C value, which is the parameter that regularize the model, as described in Section 3.1.1.3, and gamma, the kernel coefficient. Gamma represents the kernel coefficient used by the algorithm [108]. The C parameter and Gamma parameter

have the biggest impact on the algorithm performance [109]. Figure 52 shows the considered values for each of the three parameters.

```
kernel = ['rbf', 'linear', 'poly']
C = [1, 5, 20, 50, 75, 100, 500]
gamma = ['scale', 'auto']
svc_grid = dict(kernel=kernel, C=C, gamma=gamma)
```

Figure 52—SVM Grid Search

#### 4.5.2.3 Exhaustive Search

Following a similar approach to Grid Search, an exhaustive tuning of the hyperparameters was done. This tuning process used the same hyperparameter values of Grid Search. It tried every combination of the hyperparameters of the grid. This was inspired by Guyon and Elisseeff [110] and although computationally expensive, it can be done if the number hyperparameters and values are not too large.

The model was then trained and tested. Out of all the combination, the one with the highest F1-Score was kept and used to train the model.

#### 4.5.3 Model Training

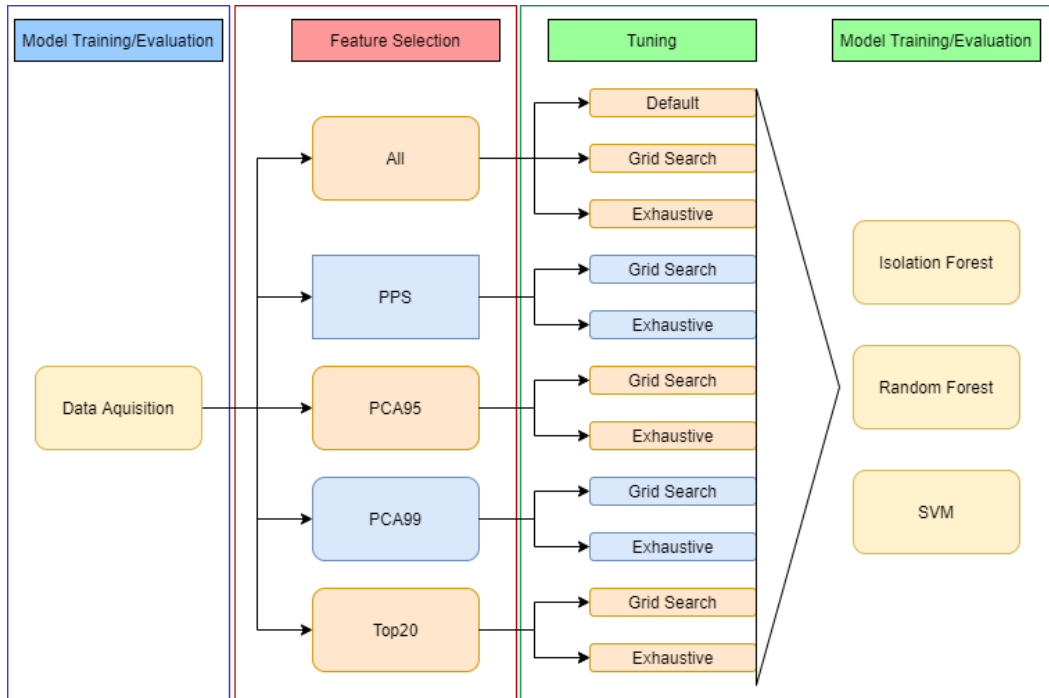
Finally, for both Velociraptor and Volatility, the three machine learning algorithms were used to trained models using the different feature reduction techniques and hyperparameter tuning mentioned in the previous sections. For each algorithm, 11 different combinations of trained models were generated, as shown by Table 12. The five feature selection techniques chosen for this thesis are all features (all), positive PPS correlation (PPS), PCA with 95% variance retained (PCA95), PCA with 99% variance retained (PCA99) and top 20 features based on *feature\_importances* (Top20). For each of those, two tuning methods were used, Grid Search or Exhaustive. A third method, default parameters, was used only once for comparison purposes with all features.

Table 12—Trained model combinations

Features	Tuning
All features	Default parameters
All features	Grid Search
All features	Exhaustive
Positive PPS correlation	Grid Search
Positive PPS correlation	Exhaustive
PCA with 95% variance retained	Grid Search
PCA with 95% variance retained	Exhaustive
PCA with 99% variance retained	Grid Search
PCA with 99% variance retained	Exhaustive
Top 20 features with <i>feature importances</i>	Grid Search
Top 20 features with <i>feature importances</i>	Exhaustive

In summary, for each of the two methodologies, each combination shown in Table 12 were used to train a machine learning model. This was done three times, once for each of the three machine learning algorithms. This mean that for each of the 2 methodology, 33 different models were trained and evaluated.

In Figure 53, the methodology presented in this section is summarized. The data was collected using Velociraptor the implemented method and Volatility for the validation method. For each method, features were generated. From this initial feature set, five different reduced feature sets were used to train a machine learning model. The hyperparameters were tuned using the default hyperparameters values, Grid Search or Exhaustive Search. Finally, each model was trained using one of the three algorithms, Isolation Forest, Random Forest and SVM. In the next section, the results of the last stage of the machine learning pipeline, results, is presented and discussed.



**Figure 53—Summary of the methodology used for both methods to train the machine learning models**



## Section 5

# Results

This section describes the results obtained from the experiment using Volatility and Velociraptor. The overall performance metrics obtained during the tests are presented, as well as the types of malware that were missed. It also provides context with regard to the results implication and how it fits with the research question.

The aim of this thesis is to propose an automated solution based on digital forensics and machine learning to detect intrusions proactively across multiple computers. The experiment was conducted with the enterprise network and collection methodology explained in Section 4.2.4. The results show that the implemented method based on Velociraptor and machine learning is effective at detecting malware compared to using Volatility and machine learning. In this section, those two methodologies are referred to as Velociraptor and Volatility, respectively.

### 5.1 Results

The results of the different trained machine learning models discussed in Figure 53 are presented. The implemented method using Velociraptor is compared, algorithm by algorithm, with the validation methodology using Volatility. Results obtained with Isolation Forest are discussed in Section 5.1.2, Random Forest in Section 5.1.3, and SVM is Section 5.1.4.

#### 5.1.1 Performance Metrics

Each model was evaluated using accuracy, precision, recall and F1-Score. Those metrics were covered in Section 2.1. Equations 1–4 provided details regarding those four metrics, respectively. This section provides more practical details about those metrics and their implications relating to this thesis.

Accuracy provides some indication of a model performance. However, on its own, it is not the most useful metric. This is due to the fact that it only represents the percentage of correctly classified samples. For imbalanced datasets such as the one used in this thesis, it is not very useful, as a classifier which would identify all the samples as normal data would still achieve a high accuracy [7].

Precision is a metric that shows how many of the samples identified as malicious are indeed malicious. It does not take into account missed malware samples. A high precision score indicates that few normal benign data has been classified as anomalous [7].

Recall improves from this as it indicates how many of the malware samples were correctly classified. A perfect classifier which has correctly identified all malware samples would have a recall value of 1. It does not indicate, however, how many false positives were generated to reach that score [7].

Finally, the F1-Score combines precision and recall to provide a more complete picture of the overall performance of the algorithm. The F1-Score is calculated using the harmonic mean of precision and recall. A high F1-Score indicates that both precision and recall values are high [7].

F1-Score was used as the main comparison metric and for the algorithm tuning as false positives need to be kept at a reasonable level in order for a classifier to be useful at detecting malware; if too many false positives are identified, it is difficult to find the actual malware samples among the benign samples. As such the best models for each algorithm were selected based on the F1-Score. Therefore, it provides a better overall assessment of the results. All four metrics are shown in the tables presenting the results from the thesis experimentation.

### 5.1.2 Isolation Forest

In this section, the results of each model trained with Isolation Forest for both Volatility and Velociraptor are presented.

The results for Volatility are shown in Table 13. In the table, each row represents a different combination of selected features and hyperparameter tuning technique, as explained in Section 4.5.3. Each of those settings were used to train a different machine learning model, using Isolation Forest. For each of the 11 models in Table 13, the accuracy, precision and recall values are displayed. The trained model with the highest F1-Score is identified in bold. Table 13 to Table 18 are all organized the same way, using their respective method and algorithm.

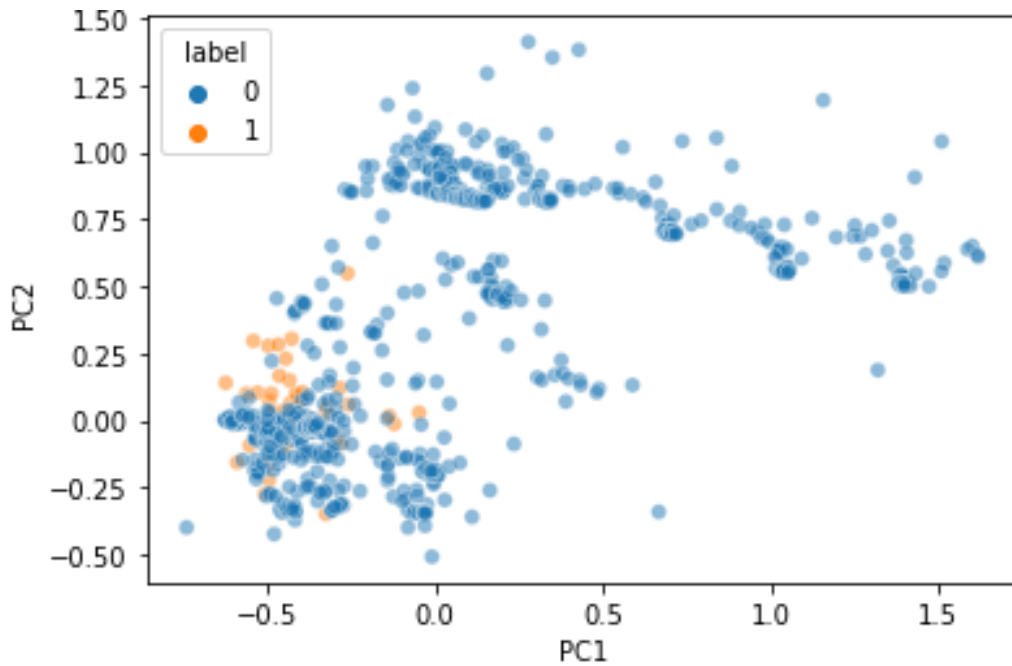
While the accuracy is relatively high for all models, the precision, recall and F1-Score are very low. As explained in Section 5.1.1, this means that while most of the data points were correctly classified as non-malicious, it did not perform well. Since the dataset is very unbalanced, a high accuracy value is irrelevant on its own. Some models had a score of 0 for precision, recall and F1-Score, such as the PCA99 tuned with Grid Search; this means that the classifier identified all the samples as benign and non-malicious, even if the accuracy value was of 0.976. This also indicates that the model could not effectively isolate the outlier data points. One of the reasons is that data points could be too closely grouped together for the Isolation Forest algorithm to effectively identify outliers.

The model which showed the best performance was trained with all features and the manual exhaustive turning method, with an F1-Score of 0.276, which is still low. The recall value of this model was of 0.216, which means that only 21.6% of the malware samples were considered anomalous. In addition, the precision value showed that only 38.1% of the samples identified as malicious were indeed malicious.

**Table 13—Results using Isolation Forest with Volatility**

Features	Tuning	Accuracy	Precision	Recall	F1-Score
All	Default	0.949	0.429	0.129	0.198
All	Grid Search	0.981	0.000	0.000	0.000
<b>All</b>	<b>Exhaustive</b>	<b>0.971</b>	<b>0.381</b>	<b>0.216</b>	<b>0.276</b>
PPS	Grid Search	0.985	0.000	0.000	0.000
PPS	Exhaustive	0.985	0.000	0.000	0.000
PCA95	Grid Search	0.984	0.000	0.000	0.000
PCA95	Exhaustive	0.969	0.143	0.100	0.118
PCA99	Grid Search	0.976	0.000	0.000	0.000
PCA99	Exhaustive	0.951	0.571	0.162	0.253
Top20	Grid Search	0.973	0.000	0.000	0.000
Top20	Exhaustive	0.969	0.143	0.100	0.118

In Figure 54, a two-dimension reduction using PCA of the Volatility entire dataset, is used to visually examine the data. The colours in the plot represent the labels of the data. The blue points represent the baseline collections, and the orange points represent the malware collections. It can be observed that the orange points are concentrated in the bottom left corner. This can make it difficult for Isolation Forest to select these points as outliers, potentially explaining its poor performance.



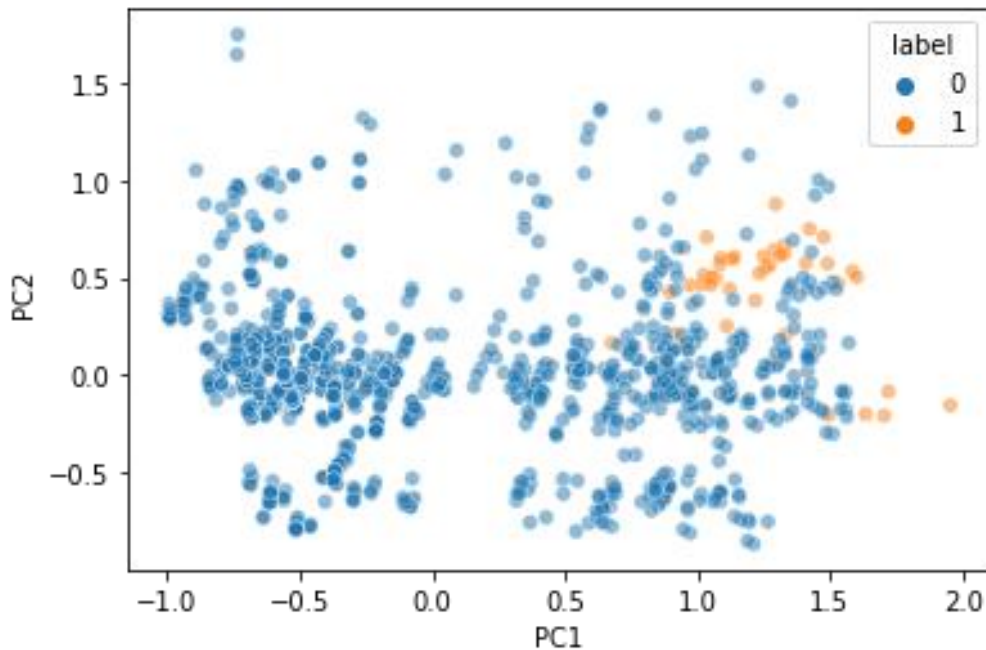
**Figure 54—PCA reduction to two dimensions of the Volatility entire dataset, with normal data points shown in blue and malware points shown in orange**

For Velociraptor, the performance of the models trained with Isolation Forest are presented in Table 14. The model that achieved the best performance was also using the top 20 features with the manual exhaustive tuning method, this time with a F1-Score of 0.844. In addition, 75% of the malware samples were found, with a precision score of 85.7%. While the overall performance is still low, it is much better than what was obtained with Volatility.

**Table 14—Results using Isolation Forest with Velociraptor**

Features	Tuning	Accuracy	Precision	Recall	F1-Score
All	Default	0.935	0.952	0.313	0.471
All	Grid Search	0.970	0.048	0.500	0.087
<b>All</b>	Exhaustive	0.987	0.857	0.750	0.800
PPS	Grid Search	0.964	0.048	0.167	0.074
PPS	Exhaustive	0.971	0.667	0.519	0.583
PCA95	Grid Search	0.964	0.000	0.000	0.000
PCA95	Exhaustive	0.961	0.286	0.333	0.308
PCA99	Grid Search	0.962	0.000	0.000	0.000
PCA99	Exhaustive	0.977	0.571	0.632	0.600
Top20	Grid Search	0.978	0.381	0.800	0.516
<b><u>Top20</u></b>	<b><u>Exhaustive</u></b>	<b><u>0.990</u></b>	<b><u>0.905</u></b>	<b><u>0.792</u></b>	<b><u>0.844</u></b>

Looking at the data separation for Velociraptor, a trend similar to Volatility is observed. In Figure 55, a two-dimension reduction using PCA of the Velociraptor entire dataset is used to visually examine the dataset. The colours in the plot represent the labels, with the blue points representing the baseline collection and the orange points representing the malware collections. The malware samples are located on the right of the plot. However, the identification of outliers would be difficult without increasing the contamination rate significantly, to a point where too many alerts would be generated.



**Figure 55—PCA reduction to two dimensions of the Velociraptor entire dataset, with normal data points shown in blue and malware points shown in orange**

For both methods, Isolation Forest was not the most effective algorithm. However, Velociraptor obtained a much higher F1-Score compared to Volatility.

### 5.1.3 Random Forest

In this section, the results of each model trained with Random Forest for both Volatility and Velociraptor are presented. In both cases, it showed more potential and better overall results for both algorithms, as shown in Table 15 and Table 16.

For Volatility, four models led to an F1-Score of 0.865: using the default settings or the manual exhaustive search with all features and with the manual exhaustive search or Grid Search with the top 20 features. Those four models were able to detect all the malware samples with 76.2% precision. The results are shown in Table 15.

It is interesting to see that the recall values are of either 1.000 or 0.000. This suggests that some overfitting could exist in the current dataset. This issue is discussed in Section 5.3.2.

**Table 15—Results using Random Forest with Volatility**

Features	Tuning	Accuracy	Precision	Recall	F1-Score
<b>All</b>	<b>Default</b>	<b>0.997</b>	<b>0.762</b>	<b>1.000</b>	<b>0.865</b>
All	Grid Search	0.996	0.714	1.000	0.833
<b>All</b>	<b>Exhaustive</b>	<b>0.997</b>	<b>0.762</b>	<b>1.000</b>	<b>0.865</b>
PPS	Grid Search	0.990	0.333	1.000	0.500
PPS	Exhaustive	0.990	0.333	1.000	0.500
PCA95	Grid Search	0.985	0.000	0.000	0.000
PCA95	Exhaustive	0.985	0.000	0.000	0.000
PCA99	Grid Search	0.985	0.000	0.000	0.000
PCA99	Exhaustive	0.986	0.048	1.000	0.091
<b>Top20</b>	<b>Grid Search</b>	<b>0.997</b>	<b>0.762</b>	<b>1.000</b>	<b>0.865</b>
<b>Top20</b>	<b>Exhaustive</b>	<b>0.997</b>	<b>0.762</b>	<b>1.000</b>	<b>0.865</b>

For Velociraptor, the same trend was visible, but was more pronounced. The recall values were also of either 1.000 or 0.000, but so were the F1-Scores. All models, except PCA, achieved 100 percent across all metrics with a perfect classification, while the four models using PCA detected none of the malware. This issue is discussed in Section 5.3.2. Table 16 shows the results for Random Forest with the Velociraptor dataset.



**Table 16—Results using Random Forest with Velociraptor**

Features	Tuning	Accuracy	Precision	Recall	F1-Score
All	Default	1.000	1.000	1.000	1.000
All	Grid Search	1.000	1.000	1.000	1.000
All	Exhaustive	1.000	1.000	1.000	1.000
PPS	Grid Search	1.000	1.000	1.000	1.000
PPS	Exhaustive	1.000	1.000	1.000	1.000
PCA95	Grid Search	0.970	0.000	0.000	0.000
PCA95	Exhaustive	0.968	0.000	0.000	0.000
PCA99	Grid Search	0.968	0.000	0.000	0.000
PCA99	Exhaustive	0.970	0.000	0.000	0.000
Top20	Grid Search	1.000	1.000	1.000	1.000
Top20	Exhaustive	1.000	1.000	1.000	1.000

Using Random Forest, the optimal models for both methods were able to detect all the malware samples as they have a recall value is of 1.000. However, both Volatility and Velociraptor showed signs of overfitting. This could be due to repetition in the dataset, as indicated in the correlation matrix and the PPS matrix in Section 4.2.5, or to simpler cross-validation method used since there were not enough malware samples in the dataset. This issue is discussed in Section 5.3.2. In terms of validation, Velociraptor outperformed Volatility and was able to detect all the malware samples.

#### 5.1.4 Support Vector Machine

In this section, the results of each model trained with SVM for both Volatility and Velociraptor are presented. Both methods were able to detect most of the malware samples, as shown in Table 17 and Table 18.

For Volatility, the best model with SVM was using all features tuned with Grid Search and manual exhaustive search. Both models led to a F1-Score of 0.857. The results are shown in Table 17. The default configuration, which was not tuned, and both models using the PPS feature reduction method, which only used one feature, had the worst results.

This is likely due to the PPS feature set only having one feature. Thus, this method was not an effective feature reduction technique for Volatility.

**Table 17—Results using SVM with Volatility**

Features	Tuning	Accuracy	Precision	Recall	F1-Score
All	Default	0.985	0.000	0.000	0.000
<b>All</b>	<b>Grid Search</b>	<b>0.996</b>	<b>0.857</b>	<b>0.857</b>	<b>0.857</b>
<b>All</b>	<b>Exhaustive</b>	<b>0.996</b>	<b>0.857</b>	<b>0.857</b>	<b>0.857</b>
PPS	Grid Search	0.985	0.000	0.000	0.000
PPS	Exhaustive	0.985	0.000	0.000	0.000
PCA95	Grid Search	0.959	0.143	0.068	0.092
PCA95	Exhaustive	0.959	0.143	0.068	0.092
PCA99	Grid Search	0.981	0.143	0.231	0.176
PCA99	Exhaustive	0.981	0.143	0.231	0.176
Top20	Grid Search	0.995	0.857	0.818	0.837
Top20	Exhaustive	0.995	0.857	0.818	0.837

For Velociraptor, four models led to a perfect classification: using all features with the manual exhaustive search, PPS with the manual exhaustive search and both methods using the top 20 features. The results are presented in Table 18. In addition, similarly to Random Forest, the recall values were of either 0.000 or 1.000. This issue is discussed in Section 5.3.2.

**Table 18—Results using SVM with Velociraptor**

<b>Features</b>	<b>Tuning</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>
All	Default	0.999	0.952	1.000	0.976
All	Grid Search	0.999	0.952	1.000	0.976
<b>All</b>	<b>Exhaustive</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
PPS	Grid Search	0.997	0.905	1.000	0.950
<b>PPS</b>	<b>Exhaustive</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
PCA95	Grid Search	0.961	0.000	0.000	0.000
PCA95	Exhaustive	0.955	0.000	0.000	0.000
PCA99	Grid Search	0.964	0.000	0.000	0.000
PCA99	Exhaustive	0.958	0.000	0.000	0.000
<b>Top20</b>	<b>Grid Search</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>
<b>Top20</b>	<b>Exhaustive</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>

Using SVM, the optimal models for both methods were able to detect most of the malware samples. However, the perfect classification is an issue and is discussed in Section 5.3.2.

The results presented in this section showed that some of the samples were not detected by the algorithms. The next section analyses which malware types were detected or missed.

## 5.2 Missed Malware

In order to determine which malware were missed by the two methods, the best trained model for each algorithm, based on the F1-Score, was reused for both Volatility and Velociraptor. Those models were identified in bold in Table 13 to

Table 18. If more than one model proved optimal, one was selected. The fitted models using those features and tuning settings were then used to classify the entire dataset, so that all 41 malware samples could be analyzed. Once all the data was classified, the name and details of each malware were appended to the results based on the hostname and

timestamp. It was then possible to analyze which malware sample and malware types were detected or missed for each method.

Table 19 shows the detection rates of Volatility for each malware types, for each algorithm. For example, for the first row, Credential Stealing malware, which has 2 samples, Isolation Forest found none of the 2 malware samples, while Random Forest and SVM found all two. Isolation Forest did underperform compared to the other two algorithms. It missed all the credential stealing, persistence and the ransomware samples. It also did poorly with the rootkit samples. This is surprising as the paper from Cohen and Nissim [4], on which most of the Volatility features were based from, had much better results. This is covered in Section 5.3.1. Only the data exfiltration and the post-exploitation tool instances were detected. Random Forest and SVM were able to detect most malware, but also had difficulties with the RATs and the rootkits. Random Forest missed both the CatfishFileSchredder and CatfishSocket1 RATs as well as three out the five instances of the hidden rootkit. SVM missed the same RATs but only missed the hidden file instance of the hidden rootkit.

**Table 19—Volatility malware type detection, by algorithms**

Malware Type	Malwares	IF Detection	RF Detection	SVM Detection
<b>Credentials Stealing</b>	2	0.00	100.00	100.00
<b>Data Exfiltration</b>	1	100.00	100.00	100.00
<b>Persistence</b>	7	0.00	100.00	100.00
<b>Post-exploitation tool</b>	3	100.00	100.00	100.00
<b>RAT</b>	21	28.57	90.48	90.48
<b>Ransomware</b>	1	0.00	100.00	100.00
<b>Rootkit</b>	6	16.67	50.00	83.33

For Velociraptor, the results are shown in Table 20. The best models for Random Forest and SVM led to 100 percent detection, while Isolation Forest missed some of the Credential Stealing malware, namely CatfishPersister ran with user-level privilege, as well

as all data exfiltration attempts and some of the RATs samples, such as CatfishPowershell1, CatfishExplorer, Cobalt Strike process injection and screenshot.

**Table 20—Velociraptor malware type detection, by algorithms**

Malware Type	Malwares	IF Detection	RF Detection	SVM Detection
<b>Credentials Stealing</b>	2	50.00	100.00	100.00
<b>Data Exfiltration</b>	1	0.00	100.00	100.00
<b>Persistence</b>	7	100.00	100.00	100.00
<b>Post-exploitation tool</b>	3	100.00	100.00	100.00
<b>RAT</b>	21	80.95	100.00	100.00
<b>Ransomware</b>	1	100.00	100.00	100.00
<b>Rootkit</b>	6	100.00	100.00	100.00

### 5.3 Discussion

The same methodology was used for both Volatility and Velociraptor in an effort to facilitate the comparison of both tools and validate the implemented methods. For each of the three algorithms, Velociraptor outperformed Volatility. However, some questions arise.

#### 5.3.1 Volatility Performance

An interesting finding was that the machine learning algorithms, using features generated from Volatility, performed poorly when trying to detect rootkits. This is surprising because the paper from Cohen and Nissim [4], on which most of the Volatility features were based on, had much better results at detecting their ransomware samples and RATs samples. In addition, Wang et al. [5], who based their work on rootkit detection on [4] also had better results on rootkit detection. While some features could not be implemented in this thesis, as mentioned in Section 4.2.3.1, a higher detection rate was expected due to the sum of the selected features from Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6].

In Table 21, the results from this thesis, from Cohen and Nissim [4] experiment with RAT and Ransomware and from Wang et al. [5] are presented. While this thesis achieved a higher recall value, the F1-Score is much lower. This is due to a higher number of false positives obtained. While all the malware samples were correctly classified, multiple samples from the baseline were misclassified.

**Table 21—Comparison of the results obtained with Volatility with the works from Cohen and Nissim [4] and Wang et al. [5]**

Features	Malware Type(s)	Algorithm	Recall	F1-Score
<b>This Thesis</b>	Multiple	RF	1.000	0.865
<b>Cohen and Nissim</b>	Ransomware	RF	0.923	0.924
<b>Cohen and Nissim</b>	RAT	RF	0.927	0.947
<b>Wang et al.</b>	Rootkit	RF	0.984	0.986

The methodology used by Cohen and Nissim [4] and Wang et al. [5] had some similarities with the method used in Section 4. However, they were some key differences between theirs and the methodology presented in this thesis.

First, Cohen and Nissim [4] used virtual machines on an ESXi Server using in vSphere. The CSTE environment used for this paper works in the same way and also has vSphere as the client-server interface. However, a key difference lies in the collection methodology. They used VM snapshots to collect the virtual memory. They used the virtual memory file from a snapshot of the virtual machine, the \*.vmsn file. From there, they extracted the volatile memory data and saved to a \*.dmp memory capture file. The idea was that even if a malware has VM evasion capabilities, it cannot interfere with the hypervisor and the memory capture [4]. However, in this thesis, a live-memory capture was performed using the Dumpit software. The Volatility executable was then run locally on the newly generated files. The resulting plugins' output, JSON files, were then exported to a server for analysis. The advantage of this method is that it is applicable to a non-virtualized environment, while snapshots can only be performed in virtualized environment.

Second, another important difference in methodology is the baseline generation. Cohen and Nissim [4] have set up a web server in their environment, so that the computers would browse websites on this server. 100 snapshots were taken, at ten-minute intervals. The last snapshot was also used as the clean state recovery point after running programs. Once the baseline was generated, they ran nine programs, four benign (Process Monitor, Wireshark, the Microsoft Windows default Disk Defragmenter utility tool and Avast anti-virus) and five malicious ransomware (Cerber, TeslaCrypt, Vipasana, Chimera and HiddenTear). For each of those nine programs, 100 more snapshots were taken before reverting to the clean state, again at ten-minute intervals. This was done so that the progression of file encryption could be detected throughout the multiple snapshots. An advantage of this methodology is that if important data was paged on the disk and not available in memory at a certain point in time, the next memory capture may be able to catch it, which mitigates this issue [4]. In total, 1000 snapshots were processed by their algorithms. This differs from the methodology used for this thesis as it involved a sophisticated user simulation using HALO to collect a robust baseline. In addition, each malware collection was done once during the malware execution, after enough time had elapsed, so that the main characteristics could be detected. This generated 41 malware samples, in comparison to the 500 samples they generated with their five malwares. Also, no benign programs were installed. Instead, the HALO agent ran multiple programs, including Microsoft Office tools, during the baseline collection as part of its profile.

A third difference is that only one memory capture for each malware was done and could have affected the performance of the implemented methodology; less malware sample makes it more difficult to perform a robust cross-validation, but having multiple samples nearly identical can be repetitive and affect performance.

Lastly, the fact that more noise was generated in the baseline generated with HALO vs the baseline generated by Cohen and Nissim [4] as they only used HTTP request; this could have impacted the strength of their methodology as their background activity was much simpler, which means their malware would stand out more and would be easier to detect.

### 5.3.2 Perfect Classification

Both Random Forest and SVM algorithms using Velociraptor features had instances with perfect classification, or very poor classification with both Random Forest and SVM. Those results are surprising and indicate some issues with the data processing and the results themselves. It can also be a sign of overfitting in the dataset.

First, Random Forest does not require scaling. Tree-based algorithms such as Random Forest and Isolation Forest do not gain additional performance using scaling. Training the dataset with a non-scaled dataset for those two algorithms could improve performance.

Second, some types of malware are under represented in the dataset. For example, there was only one ransomware and one data exfiltration tool used, but 21 RAT were used. In addition, the 19 samples generated using Cobalt Strike could have introduced some bias in the algorithm as part of the collection is similar, such as using the same tool command and control channel.

Third, the class imbalance in the dataset may have caused part of the results shown in Table 16. There were only 41 malicious samples for 1340 benign samples for Velociraptor, and 2831 samples for Volatility. This means the ratios of malicious to benign samples was 0.031 and 0.0145, respectively. Two main techniques exist to fix this issue: under-sampling the majority class, or oversampling the minority class. Since the number of malicious samples is small compared to the overall collected data, under sampling is not implementable. Over-sampling the minority class is a better option as it generates more artificial malicious samples. However, random over-sampling can lead to overfitting that benefits the minority class [111]. A hybrid approach, data augmentation can be used. It is which is a technique used to artificially increase the data that can be used in the training set. It is used to improve the generalization of the model. This technique works by generating data points that have values similar to the real data points so they look realistic [7]. It can also under-sample the majority class as well to further increase performance. The Synthetic Minority Over-sampling Technique (SMOTE) [112] is a data augmentation method that could be used in future work to improve the results and can be



implemented using `imbalanced-learn` [113], which is compatible with `SKLearn`. Using those techniques would increase the number of data points in the dataset and enable the use of a more complex cross-validation using K-fold. It would also ensure the dataset would better generalize.

Fourth, using both baseline samples and malware samples during the training could have impacted the results. Further experimentation with different dataset training, including training with only baseline data, should be performed.

Fifth, performing multiple classifications using different testing and training set and taking the overall metrics for those approaches could give a better overall view of the performance of the methodology. Using only one testing and training set is very specific and the values could change.

Finally, the perfect classification can be a sign of overfitting. Another method of reducing overfitting includes the use machine learning algorithms which uses bagging, such as bagging trees, or boosting, such as AdaBoost. This was both discussed in Section 3.1.5.1.1 and 3.1.5.1.2.

## 5.4 Validation

The aim of this thesis is to propose an automated solution based on digital forensics and machine learning to detect intrusions proactively across multiple computers. To validate this thesis, the implemented method of using `Velociraptor` and machine learning was compared to the validation method of using `Volatility` and machine learning.

The validation method was designed by reproducing the experiment conducted by similar research; features from Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6] were used to generate the validation dataset. From their research, 40 features generated from `Volatility` plugins were selected to train the machine learning models.

To be validated, the implemented method, using `Velociraptor`, needed to outperform the `Volatility` method. The best results identified in Section 5.1, for each

machine learning algorithm, are presented in Table 22. In all three cases, the implemented methodology using Velociraptor outperformed the Volatility methodology. Of note, some of the performance metrics obtained a value of 1.000. A precision of 1.000 indicates that no false positives were detected, and a recall value of 1.000 indicates that no false negative were generated.

**Table 22—Comparison of Volatility and Velociraptor, for each algorithm**

<b>Tool</b>	<b>Algorithm</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>
<b>Volatility</b>	IF	0.971	0.381	0.216	0.276
<b>Velociraptor</b>	IF	0.990	0.905	0.792	0.844
<b>Volatility</b>	RF	0.997	0.762	1.000	0.865
<b>Velociraptor</b>	RF	1.000	1.000	1.000	1.000
<b>Volatility</b>	SVM	0.996	0.857	0.857	0.857
<b>Velociraptor</b>	SVM	1.000	1.000	1.000	1.000

Table 22 shows that the implemented methodology is valid and that using Velociraptor and machine learning is an effective method which can detect malware on multiple remote computers. In addition, it requires fewer computing resources and is more transparent to the user compared to Volatility, as mentioned in Section 3.4.

The implemented method computationally less expensive at acquiring artifacts when compared to Volatility; Velociraptor does not need a memory capture to gather the data. This means that the implemented method can be applied in a physical network, while methodologies such as Cohen and Nissim [4] of using VM snapshots are limited to environment using VMs. While their approach has the advantage of generating a more trusted collection, as no malware can take control of the hypervisor from the VM, the system has to be frozen while the snapshot is taken. As their technique requires a hypervisor, it is limited to virtual environments.

## Section 6

# Conclusion

The aim of this thesis is to propose an automated solution based on digital forensics and machine learning to detect intrusions proactively across multiple computers.

In the Literature Review, a comprehensive review of current research was presented. It covered different machine algorithms and how they work, the different types of IDS, including NIDS and HIDS, enterprise monitoring solutions and memory forensic tools and techniques. Additional information was also provided on machine learning algorithms and on malware in the Background section. This thesis built upon all those research experiments, as it used digital forensic techniques to collect digital artifacts. Velociraptor, an enterprise monitoring tool, was then used to analyze, with machine learning algorithms, host-based data to detect intrusions. This thesis addressed the limitations of current methodologies, such as current research not being implementable in a classic physical enterprise network.

The implemented methodology of using Velociraptor outperformed the validation method based on Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6] research using the memory forensic tool Volatility and machine learning to detect malware. The features used in the validation methodology were trained with the same machine learning models as the implemented methodology, with the same malware samples. This validated both work and methodology, and showed that Velociraptor is an effective tool for this domain of research. Random Forest and SVM proved to be the most effective classifiers, achieving a perfect classification over the tested dataset for the implemented method, while the best model using the validation method achieved an F1-Score of 0.865, using Random Forest.

However, although effective, the use of the Velociraptor offline collector to extract the artifacts and analyzed them outside the environment with machine learning algorithms has limitations. Since the acquisition is done directly on the live host, this means the

executable could be tempered with if an attacker managed to get administrative privileges. However, the detection of the events leading to this escalated privilege would still be detectable.

Another limitation of this thesis is the small number of malware samples used reduced the strength of the classifier and made it more prone to overfitting. Solutions to this problem were discussed in Section 5.3.2 and include the use of data augmentation to artificially increase the number of malware samples.

## 6.1 Contribution

The implemented method, which used Velociraptor and machine learning, proved to be effective at detecting malware and was validated by outperforming similar research using Volatility and machine learning, such as Cohen and Nissim [4], Wang et al. [5] and Panker and Nissim [6]. As such, this thesis made the following contributions:

1. the identification of Velociraptor as an effective tool that can be used to generate features to train a machine learning model which can effectively detect malware;
2. the creation of an effective methodology to generate data;
3. the identification of features that can detect the presence of malware on an active computer; and
4. the comparison of three machine learning algorithms that can be used to find malware with this type of dataset.

## 6.2 Future Work

In the future, an implementation using the Velociraptor server, instead of the Velociraptor offline collector, could be used in a live environment to monitor and detect threats. This would be faster than the implemented methodology and would enable faster defensive countermeasures by network defenders, as they could interact with the infected host and respond directly by putting it in quarantine or killing the malicious process. The Velociraptor Server is not geared towards this type of automated analysis, therefore a new

program using the Velociraptor API would be needed to query and process the feature. The detection model could still be trained using the methodology presented in this thesis.

In addition, this thesis only used a small number of malware samples to generate its dataset. In order to improve from this successful proof of concept, more malware samples would be needed through either acquisition of more malware binaries, or through data augmentation, so more cross-validation methods could be used, as discussed in Section 5.3.2. Other machine learning and deep learning algorithms could also be used to improve performance.

In conclusion, this thesis successfully implemented a methodology to collect disk-based and volatile artifacts using Velociraptor and process them using machine learning algorithms in order to classify the collected data. It also proved the research hypothesis that using both types of artifacts, disk-based and memory-based artifacts, provides a better view over the activities currently ongoing on the computer compared to only one type.

## References

- [1] Securus360, 'EDR is great, but its Limitations can leave you open to Cyberattacks'. <https://www.securus360.com/blog/edr-is-great-but-its-limitations-can-leave-you-open-to-cyberattacks> (accessed Apr. 08, 2022).
- [2] M. Murthaja, B. Sahayanathan, A. N. T. S. Munasinghe, D. Uthayakumar, L. Rupasinghe, and A. Senarathne, 'An Automated Tool for Memory Forensics', in *2019 International Conference on Advancements in Computing (ICAC)*, Dec. 2019, pp. 1–6. doi: 10.1109/ICAC49085.2019.9103416.
- [3] Periyadi, G. A. Mutiara, and R. Wijaya, 'Digital forensics random access memory using live technique based on network attacked', in *2017 5th International Conference on Information and Communication Technology (ICoICT7)*, May 2017, pp. 1–6. doi: 10.1109/ICoICT.2017.8074695.
- [4] A. Cohen and N. Nissim, 'Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory', *Expert Syst. Appl.*, vol. 102, pp. 158–178, Jul. 2018, doi: 10.1016/j.eswa.2018.02.039.
- [5] X. Wang *et al.*, 'TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis', *Math. Biosci. Eng.*, vol. 16, no. 4, pp. 2650–2667, 2019, doi: 10.3934/mbe.2019132.
- [6] T. Panker and N. Nissim, 'Leveraging malicious behavior traces from volatile memory using machine learning methods for trusted unknown malware detection in Linux cloud environments', *Knowl.-Based Syst.*, vol. 226, p. 107095, Aug. 2021, doi: 10.1016/j.knosys.2021.107095.
- [7] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, 2nd ed. Sebastopol, CA: O'Reilly Media, Inc, 2019.
- [8] F. M. Dekking, C. Kraaikamp, H. P. Lopuha, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Netherlands: Springer-Verlag, 2005.
- [9] B. Lachine, 'Machine Learning Introduction', Kingston, Ontario, Oct. 13, 2020. Accessed: Oct. 13, 2021. [Online]. Available: <https://moodle.rmc.ca>
- [10] D. Spiekermann and J. Keller, 'Unsupervised packet-based anomaly detection in virtual networks', *Comput. Netw.*, vol. 192, p. 108017, Jun. 2021, doi: 10.1016/j.comnet.2021.108017.
- [11] E. Aghaei and G. Serpen, 'Host-based anomaly detection using Eigentraces feature extraction and one-class classification on system call trace data', *J. Inf. Assur. Secur.*, vol. 14, no. 4, p. 11, 2019.

- [12] R. M. A. Mohammad and M. Alqahtani, 'A comparison of machine learning techniques for file system forensics analysis', *J. Inf. Secur. Appl.*, vol. 46, pp. 53–61, Jun. 2019, doi: 10.1016/j.jisa.2019.02.009.
- [13] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, 'Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains', *Lead. Issues Inf. Warf. Secur. Res.*, vol. 1, no. 1, p. 14, 2011.
- [14] S. Jose, D. Malathi, B. Reddy, and D. Jayaseeli, 'A Survey on Anomaly Based Host Intrusion Detection System', *J. Phys. Conf. Ser.*, vol. 1000, p. 012049, Apr. 2018, doi: 10.1088/1742-6596/1000/1/012049.
- [15] P. Pols, 'Unified Kill Chain: Raising Resilience Against Cyber Attacks', White Paper. Accessed: Apr. 24, 2022. [Online]. Available: <https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf>
- [16] C. Yucel, A. Lockett, I. Chalkias, D. Mallis, and V. Katos, 'MAIT: Malware Analysis and Intelligence Tool', *Inf. Secur. Int. J.*, vol. 50, pp. 49–65, 2021, doi: 10.11610/isij.5024.
- [17] J. E. Thomas, 'Individual Cyber Security: Empowering Employees to Resist Spear Phishing to Prevent Identity Theft and Ransomware Attacks', *Int. J. Bus. Manag.*, vol. 13, no. 6, p. 1, Apr. 2018, doi: 10.5539/ijbm.v13n6p1.
- [18] C. Zimmerman, *Ten Strategies of a World-Class Cybersecurity Operations Center*. McLean, VA: The MITRE Corporation, 2014.
- [19] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. (Guenevere) Chen, 'A Survey of Intrusion Detection Systems Leveraging Host Data', *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–35, Jan. 2020, doi: 10.1145/3344382.
- [20] J. P. Anderson, 'Computer Security Threat Monitoring and Surveillance', Technical Report, James P. Anderson Company, Fort Washington, Pa, 79F296400, 1980.
- [21] T. Lunt, 'Automated Audit Trail Analysis and Intrusion Detection: A Survey', in *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, 1998-10 1988, vol. 353, p. 11.
- [22] T. F. Lunt *et al.*, 'A Real-Time Intrusion-Detection Expert System', SRI International, Washington, DC, SRI Project 6784, 1992.
- [23] M. Sikorski and A. Honig, *Practical malware analysis: The hands-on guide to dissecting malicious software*. San Francisco, CA: No Starch Press, 2012.
- [24] M. Christodorescu and S. Jha, 'Static Analysis of Executables to Detect Malicious Patterns', in *Proceedings of the 12th USENIX Security Symposium (Security'03)*, Washington, D.C., Aug. 2003, pp. 169–186. Accessed: May 09, 2022. [Online]. Available: <https://courses.cs.cornell.edu/cs711/2005fa/papers/cj-usenix03.pdf>

- [25] A. Patcha and J.-M. Park, ‘An overview of anomaly detection techniques: Existing solutions and latest technological trends’, *Comput. Netw.*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007, doi: 10.1016/j.comnet.2007.02.001.
- [26] S. More, M. Matthews, A. Joshi, and T. Finin, ‘A Knowledge-Based Approach to Intrusion Detection Modeling’, in *2012 IEEE Symposium on Security and Privacy Workshops*, May 2012, pp. 75–81. doi: 10.1109/SPW.2012.26.
- [27] The MITRE Corporation, ‘CVE’. <https://cve.mitre.org/> (accessed Apr. 25, 2022).
- [28] M. T. R. Laskar *et al.*, ‘Extending Isolation Forest for Anomaly Detection in Big Data via K-Means’, *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 4, pp. 1–26, Oct. 2021, doi: 10.1145/3460976.
- [29] M. Stampar and K. Fertalj, ‘Artificial intelligence in network intrusion detection’, in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2015, pp. 1318–1323. doi: 10.1109/MIPRO.2015.7160479.
- [30] R. V. Mendonça *et al.*, ‘Intrusion Detection System Based on Fast Hierarchical Deep Convolutional Neural Network’, *IEEE Access*, vol. 9, pp. 61024–61034, 2021, doi: 10.1109/ACCESS.2021.3074664.
- [31] A. Qayyum, M. H. Islam, and M. Jamil, ‘Taxonomy of statistical based anomaly detection techniques for intrusion detection’, in *Proceedings of the IEEE Symposium on Emerging Technologies, 2005.*, Islamabad, Pakistan, 2005, pp. 270–276. doi: 10.1109/ICET.2005.1558893.
- [32] V. A. Siris and F. Papagalou, ‘Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks’, *IEEE Glob. Telecommun. Conf. 2004 GLOBECOM04*, vol. 4, pp. 2050–2054, 2004.
- [33] C.-Y. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt, ‘A specification-based intrusion detection system for AODV’, in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks - SASN '03*, Fairfax, Virginia, 2003, p. 125. doi: 10.1145/986858.986876.
- [34] Q. Liu, V. Hagenmeyer, and H. B. Keller, ‘A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids’, *IEEE Access*, vol. 9, pp. 57542–57564, 2021, doi: 10.1109/ACCESS.2021.3071263.
- [35] G. Karantzas and C. Patsakis, ‘An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors’, *J. Cybersecurity Priv.*, vol. 1, no. 3, pp. 387–421, Jul. 2021, doi: 10.3390/jcp1030021.
- [36] The MITRE Corporation, ‘System Binary Proxy Execution: Control Panel’. <https://attack.mitre.org/techniques/T1218/002/> (accessed Apr. 25, 2022).
- [37] Microsoft Corporation, ‘Memory-Mapped Files’. <https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files> (accessed Apr. 25, 2022).
- [38] M. H. Ligh, *The Art of Memory Forensics*. Indianapolis, IN: Wiley, 2014.



- [39] TWC Security, ‘DumpIt’, *GitHub*. <https://github.com/thimbleweed/All-In-USB> (accessed Apr. 04, 2022).
- [40] A. Afreen, M. Aslam, and S. Ahmed, ‘Analysis of Fileless Malware and its Evasive Behavior’, in *2020 International Conference on Cyber Warfare and Security (ICCWS)*, Islamabad, Pakistan, Oct. 2020, pp. 1–8. doi: 10.1109/ICCWS48432.2020.9292376.
- [41] A. M. A. Hameed, M. Daley, and L. Espinosa-Anke, ‘A Machine Learning Approach for Memory Forensic Investigation’, Cardiff University, 2020.
- [42] F. T. Liu, K. M. Ting, and Z.-H. Zhou, ‘Isolation Forest’, in *2008 Eighth IEEE International Conference on Data Mining*, Dec. 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.
- [43] S. Sharma, ‘Efficient log analysis using advanced detection and filtering techniques’, National College of Ireland, 2019. [Online]. Available: <http://norma.ncirl.ie/3894/1/suyashsharma.pdf>
- [44] T. K. Ho, ‘Random decision forests’, in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug. 1995, vol. 1, pp. 278–282 vol.1. doi: 10.1109/ICDAR.1995.598994.
- [45] T. Yiu, ‘Understanding Random Forest: How the Algorithm Works and Why it Is So Effective’, *Towards Data Science*, Jun. 12, 2019. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2> (accessed Mar. 23, 2022).
- [46] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, ‘Random Forests and Decision Trees’, *Int. J. Comput. Sci. Issues IJCSI*, vol. 9, no. 5, p. 7, 2012.
- [47] Statistics Canada, ‘Data Exploration: Types of variables’, Sep. 02, 2021. <https://www150.statcan.gc.ca/n1/edu/power-pouvoir/ch8/5214817-eng.htm> (accessed Apr. 21, 2022).
- [48] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, ‘A comprehensive survey on support vector machine classification: Applications, challenges and trends’, *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020, doi: 10.1016/j.neucom.2019.10.118.
- [49] A. Zisserman, ‘The SVM classifier’, Oxford University, 2015. Accessed: Apr. 21, 2022. [Online]. Available: <https://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf>
- [50] S. Khalid, T. Khalil, and S. Nasreen, ‘A survey of feature selection and feature extraction techniques in machine learning’, *Proc. 2014 Sci. Inf. Conf. SAI 2014*, pp. 372–378, Oct. 2014, doi: 10.1109/SAI.2014.6918213.
- [51] Scikit-Learn, ‘Extra Trees Classifier’, *scikit-learn*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html> (accessed Apr. 26, 2022).

- [52] Z. Luna, 'Feature Selection in Machine Learning: Correlation Matrix | Univariate Testing | RFECV', *Geek Culture*, Aug. 10, 2021. <https://medium.com/geekculture/feature-selection-in-machine-learning-correlation-matrix-univariate-testing-rfecv-1186168fac12> (accessed Mar. 24, 2022).
- [53] J. H. Moedjahedy and G. Pramudya, 'Student Achievement Classification using Power Predictive Score with Machine Learning', in *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)*, Oct. 2021, pp. 1–6. doi: 10.1109/ICORIS52787.2021.9649467.
- [54] F. Wetschoreck, 'RIP correlation. Introducing the Predictive Power Score', *Towards Data Science*, May 04, 2020. <https://towardsdatascience.com/rip-correlation-introducing-the-predictive-power-score-3d90808b9598> (accessed Mar. 17, 2022).
- [55] Kaggle, 'Titanic - Machine Learning from Disaster'. <https://kaggle.com/competitions/titanic> (accessed Mar. 24, 2022).
- [56] X. Ying, 'An Overview of Overfitting and its Solutions', *J. Phys. Conf. Ser.*, vol. 1168, no. 2, p. 022022, Feb. 2019, doi: 10.1088/1742-6596/1168/2/022022.
- [57] P. Probst, M. Wright, and A.-L. Boulesteix, 'Hyperparameters and Tuning Strategies for Random Forest', *WIREs Data Min. Knowl. Discov.*, vol. 9, no. 3, May 2019, doi: 10.1002/widm.1301.
- [58] EliteDataScience, 'Overfitting in Machine Learning: What It Is and How to Prevent It', *EliteDataScience*, Sep. 07, 2017. <https://elitedatascience.com/overfitting-in-machine-learning> (accessed Apr. 08, 2022).
- [59] L. Yang and A. Shami, 'On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice', *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020, doi: 10.1016/j.neucom.2020.07.061.
- [60] Scikit-Learn, 'Random Forest Classifier', *scikit-learn*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> (accessed Apr. 27, 2022).
- [61] P. Liashchynskiy and P. Liashchynskiy, 'Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS', *Comput. Res. Repos.*, vol. abs/1912.06059, Dec. 2019, Accessed: Mar. 28, 2022. [Online]. Available: <http://arxiv.org/abs/1912.06059>
- [62] J. Bergstra and Y. Bengio, 'Random Search for Hyper-Parameter Optimization', *J. Mach. Learn. Res.*, vol. 13, no. 2, p. 25, 2012.
- [63] J. Šafránková, Ed., 'Data Splitting', in *Proceedings of Contributed Papers*, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic., 2010, vol. Part 1, pp. 31–36.

- [64] Y. Bai *et al.*, ‘How Important is the Train-Validation Split in Meta-Learning?’, in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021, pp. 543–553. Accessed: Mar. 28, 2022. [Online]. Available: <https://proceedings.mlr.press/v139/bai21a.html>
- [65] S. Arlot and A. Celisse, ‘A survey of cross-validation procedures for model selection’, *Stat. Surv.*, vol. 4, pp. 40–79, Jan. 2010, doi: 10.1214/09-SS054.
- [66] Y. Jung, ‘Multiple predicting K-fold cross-validation for model selection’, *J. Nonparametric Stat.*, vol. 30, no. 1, pp. 197–215, Nov. 2017, doi: 10.1080/10485252.2017.1404598.
- [67] Scikit-Learn, ‘Cross-validation: evaluating estimator performance’, *scikit-learn*. [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html) (accessed Mar. 28, 2022).
- [68] W. Jiang, X. Wu, X. Cui, and C. Liu, ‘A Highly Efficient Remote Access Trojan Detection Method’, *Int. J. Digit. Crime Forensics*, vol. 11, no. 4, pp. 1–13, Oct. 2019, doi: 10.4018/IJDCF.2019100101.
- [69] Rapid7, ‘Metasploit: The world’s most used penetration testing framework’, *Metasploit*. <https://www.metasploit.com/> (accessed Mar. 29, 2022).
- [70] P. Kim, *The hacker playbook 3: practical guide to penetration testing*, Red team edition. United States: Secure Planet LLC, 2018.
- [71] Offensive Security, ‘Backdooring EXE Files | Offensive Security’. <https://www.offensive-security.com/metasploit-unleashed/backdooring-exe-files/> (accessed Mar. 29, 2022).
- [72] Infosec, ‘Rootkits: User Mode’, *Infosec Resources*. <https://resources.infosecinstitute.com/topic/rootkits-user-mode-kernel-mode-part-1/> (accessed Mar. 31, 2022).
- [73] J. Joy, A. John, and J. Joy, ‘Rootkit detection mechanism: A survey’, in *Advances in parallel distributed computing*, Berlin, Heidelberg, 2011, pp. 366–374.
- [74] Infosec, ‘Rootkits: Kernel Mode’, *Infosec Resources*. <https://resources.infosecinstitute.com/topic/rootkits-user-mode-kernel-mode-part-2/> (accessed Mar. 31, 2022).
- [75] Microsoft Corporation, ‘Driver Signing’. <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing> (accessed Apr. 26, 2022).
- [76] A. Tandon and A. Nayyar, ‘A comprehensive survey on ransomware attack: A growing havoc cyberthreat’, in *Data management, analytics and innovation*, Singapore, 2019, pp. 403–420.
- [77] Sudhakar and S. Kumar, ‘An emerging threat Fileless malware: a survey and research challenges’, *Cybersecurity*, vol. 3, no. 1, p. 1, Dec. 2020, doi: 10.1186/s42400-019-0043-x.

- [78] C. Wueest and H. Anand, ‘Living off the land and fileless attack techniques’, Symantec, Internet Security Threat Report, 2017.
- [79] M. U. Rana, M. Ali Shah, and O. Ellahi, ‘Malware Persistence and Obfuscation: An Analysis on Concealed Strategies’, in *Proceedings of the 26th International Conference on Automation & Computing*, Portsmouth, UK, Sep. 2021, pp. 1–6. doi: 10.23919/ICAC50006.2021.9594197.
- [80] L. Dubey, ‘Identifying The Malware Persistence Using Advance Static And Advance Dynamic Method’, *Int. J. Sci. Technol. Res.*, vol. 8, no. 10, p. 8, 2019.
- [81] Penetration Testing Lab, ‘Persistence – AppInit DLLs’, *Penetration Testing Lab*, Jan. 07, 2020. <https://pentestlab.blog/2020/01/07/persistence-appinit-dlls/> (accessed Mar. 31, 2022).
- [82] Microsoft Corporation, ‘Dynamic-Link Library Search Order’, Jul. 28, 2021. <https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order> (accessed Mar. 31, 2022).
- [83] The MITRE Corporation, ‘Hijack Execution Flow: DLL Side-Loading, Sub-technique T1574.002 - Enterprise | MITRE ATT&CK®’. <https://attack.mitre.org/techniques/T1574/002/> (accessed Apr. 26, 2022).
- [84] Infosec, ‘Common malware persistence mechanisms’, *Infosec Resources*, Jun. 12, 2016. <https://resources.infosecinstitute.com/topic/common-malware-persistence-mechanisms/> (accessed Mar. 31, 2022).
- [85] Rapid7, ‘Velociraptor - Dig deeper!’ <https://docs.velociraptor.app/> (accessed Feb. 01, 2021).
- [86] Jupyter, ‘Project Jupyter’. <https://www.jupyter.org> (accessed Mar. 01, 2021).
- [87] M. Cohen, ‘Velociraptor Post-Processing with Jupyter Notebook and Pandas’, Mar. 05, 2020. <https://medium.com/velociraptor-ir/velociraptor-post-processing-with-jupyter-notebook-and-pandas-8a344d05ee8c> (accessed Mar. 01, 2021).
- [88] Microsoft Corporation, ‘Event Tracing for Windows (ETW) - Windows drivers’. <https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-> (accessed Apr. 25, 2022).
- [89] M. Cohen, ‘Digging into process memory’, *Medium*, Apr. 16, 2021. <https://velociraptor.velocidex.com/digging-into-process-memory-33c60a640cdb> (accessed Aug. 22, 2021).
- [90] Field Effect, ‘Field Effect: The most sophisticated cyber threat monitoring on the planet, made simple’, *Field Effect Software Inc.* <https://fieldeffect.com/> (accessed Mar. 21, 2022).
- [91] V. Rose, A. Rose, and N. Knight, *Starkiller*. BC Security, 2022. Accessed: Mar. 29, 2022. [Online]. Available: <https://github.com/BC-SECURITY/Starkiller>

- [92] Cobalt Strike, ‘Cobalt Strike: Software for Adversary Simulations and Red Team Operations’. <https://www.cobaltstrike.com/> (accessed Mar. 29, 2022).
- [93] Cobalt Strike, ‘Screenshots’. <https://www.cobaltstrike.com/screenshots/> (accessed Mar. 29, 2022).
- [94] B. Delphy, *mimikatz*. Parrot Security (mirror of <https://nest.parrotsec.org>), 2022. Accessed: Mar. 30, 2022. [Online]. Available: <https://github.com/ParrotSec/mimikatz>
- [95] M. Fischer, *Living Off The Land*. 2022. Accessed: Apr. 04, 2022. [Online]. Available: <https://github.com/bytecode77/living-off-the-land>
- [96] Microsoft Corporation, ‘Audit SAM (Windows 10) - Windows security’, *Audit SAM*, Mar. 05, 2022. <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/audit-sam> (accessed Jun. 14, 2022).
- [97] Process Hacker, *Processhacker*. Process Hacker, 2022. Accessed: Apr. 08, 2022. [Online]. Available: <https://github.com/processhacker/processhacker>
- [98] R. Ancarani, *Offensiveph*. 2021. Accessed: Apr. 04, 2022. [Online]. Available: <https://github.com/RiccardoAncarani/OffensivePH>
- [99] M. Fischer, *r77 Rootkit*. 2021. Accessed: Nov. 04, 2021. [Online]. Available: <https://github.com/bytecode77/r77-rootkit>
- [100] J. Kornev, *Hidden*. 2022. Accessed: Apr. 04, 2022. [Online]. Available: <https://github.com/JKornev/hidden>
- [101] hasherezade, *hollows\_hunter*. 2022. Accessed: Apr. 05, 2022. [Online]. Available: [https://github.com/hasherezade/hollows\\_hunter](https://github.com/hasherezade/hollows_hunter)
- [102] M. Cohen, ‘Velociraptor Artifact to apply Hollows Hunter on the endpoint’, *Gist*. <https://gist.github.com/scudette/0f5d5102b6e3b1580b4feccdf7d59b53> (accessed Apr. 05, 2022).
- [103] technopersistent, ‘Compiling an up-to-date Standalone EXE of Volatility 2.6 in Windows’, *Tech Babble*, Feb. 09, 2020. <https://techbabble.tech.blog/2020/02/09/compiling-an-up-to-date-standalone-exe-of-volatility-2-6-in-windows/> (accessed Apr. 04, 2022).
- [104] M. Cohen, ‘Triage with Velociraptor: Part 4’, *Medium*, Jul. 13, 2020. <https://velociraptor.velocidex.com/triage-with-velociraptor-pt-4-cf0e60810d1e> (accessed Mar. 21, 2022).
- [105] A. Shetye, ‘Feature Selection with sklearn and Pandas’, *Medium*, Feb. 12, 2019. <https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b> (accessed Apr. 26, 2022).
- [106] P. Paolo, ‘Feature Selection Techniques’, *GitHub*, Oct. 03, 2019. <https://pierpaolo28.github.io//blog/blog27/> (accessed Apr. 26, 2022).

- [107] Scikit-Learn, ‘Isolation Forest’, *scikit-learn*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html> (accessed Apr. 27, 2022).
- [108] Scikit-Learn, ‘Support Vector Classification’, *scikit-learn*. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html> (accessed Apr. 27, 2022).
- [109] Scikit-Learn, ‘Support Vector Machines’, *scikit-learn*. <https://scikit-learn.org/stable/modules/svm.html> (accessed Apr. 27, 2022).
- [110] I. Guyon and A. Elisseeff, ‘An Introduction to Variable and Feature Selection’, *J. Mach. Learn. Res.*, vol. 3, no. March, p. 26, 2003.
- [111] F. López, ‘Class Imbalance: Random Sampling and Data Augmentation with Imbalanced-Learn’, *Medium*, Feb. 21, 2021. <https://towardsdatascience.com/class-imbalance-random-sampling-and-data-augmentation-with-imbalanced-learn-63f3a92ef04a> (accessed Jun. 13, 2022).
- [112] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, ‘SMOTE: Synthetic Minority Over-sampling Technique’, *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.
- [113] G. Lemaitre, F. Nogueira, and C. Aridas, ‘Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning’, *J. Mach. Learn. Res.*, vol. 18, no. 17, p. 5, 2017.

# Annex A

## Velociraptor Features List

**Table A-1—Complete Velociraptor features list**

Type	Feature	Description	Data Type	Plugin
API	ProcessHollowing_suspicious_count	Number of suspicious processes found by Hollow Hunter	Int	Custom.Windows.Detection.ProcessHollowing
DLL	num_Detection_ForwardedImports	Number of dlls which are named the same as the DLL they are forwarding to	Int	Windows.Detection.ForwardedImports
DLL	mean_dll_path_len	Mean length of the dll paths	Float	Windows.System.DLLs
DLL	max_dll_path_len	Max length of the dll paths	Int	Windows.System.DLLs
DLL	median_dll_path_len	Median length of the dll paths	Float	Windows.System.DLLs
DLL	mean_dll_entries	Mean number of dll per process	Float	Windows.System.DLLs
DLL	max_dll_dll_entries	Max number of dll per process	Int	Windows.System.DLLs
DLL	median_dll_dll_entries	Median number of dll per process	Float	Windows.System.DLLs
Events	num_unique_ProcessName_EventLogs_AlternateLogon	Number of logons specifying alternate credentials	Int	Windows.EventLogs.AlternateLogon
Events	num_unique_TargetUserName_EventLogs_AlternateLogon	Number of unique Target User Name in the AlternateLogon EventLog	Int	Windows.EventLogs.AlternateLogon
Events	num_unique_TargetServerName_EventLogs_AlternateLogon	Number of unique Target Server Name in the AlternateLogon EventLog	Int	Windows.EventLogs.AlternateLogon
Events	num_ps_severity	Number of unique levels of severity	Int	Windows.EventLogs.PowershellModule
Events	num_ps_cmd_type	Number of unique type of commands	Int	Windows.EventLogs.PowershellModule
Events	num_ps_cmd_name	Number of unique command name	Int	Windows.EventLogs.PowershellModule
Events	num_ps_User	Number of unique users	Int	Windows.EventLogs.PowershellModule

<b>Events</b>	num_ps_host_applications	Number of host application	Int	Windows.EventLogs.PowershellModule
<b>Events</b>	num_type_logon_RDPAuth	Number of unique logon type	Int	Windows.EventLogs.RDPAuth
<b>Events</b>	num_unique_EventID	Number of unique event ID	Int	Windows.EventLogs.RDPAuth
<b>Events</b>	num_unique_users	Number of unique users	Int	Windows.EventLogs.RDPAuth
<b>Events</b>	num_unique_tasks_EventLogs_ScheduledTasks	Number of unique schedule tasks	Int	Windows.EventLogs.ScheduledTasks
<b>Events</b>	num_serviceCreationComspec	Number of hits on the string "COMSPEC" in Windows Service Creation events.	Int	Windows.EventLogs.ServiceCreationComspec/ServiceCreation
<b>File System</b>	client_user_num	Total number of users on the workstation	Int	Generic.Client.Info/Users
<b>File System</b>	user_logged_on_today	Number of users that logged on today	Int	Generic.Client.Info/Users
<b>File System</b>	Atime_modified_30min	Number of files Accessed in the 30 min before the collection	Int	Generic.Forensic.Timeline
<b>File System</b>	Mtime_modified_30min	Number of files Modified in the 30 min before the collection	Int	Generic.Forensic.Timeline
<b>File System</b>	Ctime_modified_30min	Number of files Created in the 30 min before the collection	Int	Generic.Forensic.Timeline
<b>File System</b>	num_downloads	Number of downloaded files, based on the Zone.Identifier alternate data stream	Int	Windows.Analysis.EvidenceOfDownload
<b>File System</b>	num_executables	Number of downloaded files that are of an executable format	Int	Windows.Analysis.EvidenceOfDownload
<b>File System</b>	macrofound	Number of Office macros found by Velociraptor	Int	Windows.Applications.OfficeMacros.csv
<b>File System</b>	num_Detection_BinaryRename	Number of renamed binaries commonly abused by adversaries detected by Velociraptor	Int	Windows.Detection.BinaryRename
<b>File System</b>	num_Detection_CryptnetUrlCache	Number of detected downloads using Certutil	Int	Windows.Detection.CryptnetUrlCache



<b>File System</b>	num_Detection_TemplateInjection	Number of injected templates in Office documents	Int	Windows.Detection.TemplateInjection
<b>File System</b>	Mean_Forensics_Bam	Mean time between program start	Float	Windows.Forensics.Bam
<b>File System</b>	num_Forensics_CertUtil	Number of entries in the CertUtils Cache of downloaded files	Int	Windows.Forensics.CertUtil
<b>File System</b>	num_abnormal_lnk_path	Number of abnormal paths for the link file	Int	Windows.Forensics.Lnk
<b>File System</b>	count_lnk_to_remote	Number of Lnk file pointing to a different host	Int	Windows.Forensics.Lnk
<b>File System</b>	num_Forensics_RecentApps	Number of entries in the Recent App plugin	Int	Windows.Forensics.RecentApps
<b>File System</b>	bam_lateral_movement	Number of entries in Artifact.Windows.Forensics.Bam with wmic.exe as the binary	Int	Windows.Packs.LateralMovement/BAM
<b>File System</b>	wmic_lateral_movement	Number of entries in Artifact.Windows.Forensics.Prefetche with wmic.exe as the binary	Int	Windows.Packs.LateralMovement/WMIC
<b>Network</b>	URL_num	number of URL	Int	Generic.Forensic.Carving.URLs
<b>Network</b>	max_URL_entropy	Calculate the entropy of all URL found in the cache and return the maximum value	Int	Generic.Forensic.Carving.URLs
<b>Network</b>	max_SRUM_exfill	Largest NetworkBytesRaw value per millisecond	Float	Windows.Forensics.SRUM/Execution Stats
<b>Network</b>	mean_SRUM_exfill	Mean NetworkBytesRaw value per millisecond	Float	Windows.Forensics.SRUM/Execution Stats
<b>Network</b>	net_conn_amount	Number of network connection	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	tcp_conn_amount	Number of TCP connection	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	udp_conn_amount	Number of UDP connection	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	listen_amount	Number of LISTEN connection status	Int	Windows.Network.NetstatEnriched/Netstat

<b>Network</b>	established_amount	Number of ESTABLISHED connection status	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	close_wait_amount	Number of CLOSE_WAIT connection status	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	close_amount	Number of CLOSE connection status	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	different_tcp_port	Number of unique TCP port	Int	Windows.Network.NetstatEnriched/Netstat
<b>Network</b>	different_udp_port	Number of unique UDP port	Int	Windows.Network.NetstatEnriched/Netstat
<b>Registry</b>	avg_child_process	Mean number of child process per process	Float	Generic.System.Pstree
<b>Registry</b>	max_child_process	Max number of child process per process	Int	Generic.System.Pstree
<b>Registry</b>	median_child_process	Median number of child process per process	Float	Generic.System.Pstree
<b>Registry</b>	svchost_not_services	Number of svchost process where the parent process is not services.exe	Int	Generic.System.Pstree
<b>Registry</b>	num_Attack_ParentProcess	Number of executables that are mapped by Velociraptor to the MITRE Att&ck framework	Int	Windows.Attack.ParentProcess
<b>Registry</b>	num_Detection_Impersonation	Number of threads on the system which have an impersonation token	Int	Windows.Detection.Impersonation
<b>Registry</b>	num_Mutants_Handles	Number of mutant found on the system	Int	Windows.Detection.Mutants/Handles
<b>Registry</b>	max_Mutants_Handles	Maximum number of mutants found in a process	Int	Windows.Detection.Mutants/Handles
<b>Registry</b>	mean_Mutants_Handles	Mean number of mutants in all processes	float	Windows.Detection.Mutants/Handles
<b>Registry</b>	amcache_lateral_movement	Number of entries in Artifact.Windows.System.Amcache with wmic.exe as the binary	Int	Windows.Packs.LateralMovement/AmCache
<b>Registry</b>	shimcache_lateral_movement	Number of entries in Artifact.Windows.Registry.AppCompatCache with wmic.exe as the binary	Int	Windows.Packs.LateralMovement/ShimCache

<b>Registry</b>	Bootstrapping_lateral_movement	Number of programs with a debugger registry key	Int	Windows.Packs.Persistence/Debug Bootstrapping
<b>Registry</b>	num_enabled_startup	Number of Startup items that is not desktop.ini	Int	Windows.Packs.Persistence/Startup Items
<b>Registry</b>	num_wmi_event_filter_startup	Number of permanent event listener within WMI	Int	Windows.Persistence.PermanentWMIEvents
<b>Registry</b>	num_powershell_reg	Number of signatures related to general PowerShell execution in the user's profile registry hive	Int	Windows.Persistence.PowershellRegistry
<b>Registry</b>	num_wow64cpu	Number of wow64cpu.dll replacement Autorun in Windows 10	Int	Windows.Persistence.Wow64cpu
<b>Registry</b>	enabled_macro	Number entries in the registry key indicating macro was enabled by user	Int	Windows.Registry.EnabledMacro
<b>Registry</b>	num_Registry_WDigest	WDigest registry values on the filesystem	Int	Windows.Registry.Wdigest
<b>Registry</b>	num_AppcompatShims	Number of application compatibility cache entries	Int	Windows.Sys.AppcompatShims
<b>Registry</b>	pipe_amount	Number of pipes in use on the system	Int	Windows.System.Handles
<b>Registry</b>	num_proc_using_pipes	Number of Processes with named pipes	Int	Windows.System.Handles
<b>Registry</b>	num_pipes_names	Number of unique pipe names	Int	Windows.System.Handles
<b>Registry</b>	num_pipes_handles	Number of unique pipe handles	Int	Windows.System.Handles
<b>Registry</b>	num_untrusted_binaries	Number of common systems binaries that are signed and that the signature does not match	Int	Windows.System.UntrustedBinaries