Cape Peninsula
University of Technology

ANT TREE MINER AMYNTAS FOR INTRUSION DETECTION

by

FRANS HENDRIK BOTES

Thesis submitted in fulfilment of the requirements for the degree

Master of Technology: Information Technology

in the Faculty of Informatics and Design

at the Cape Peninsula University of Technology

Promoter: Prof. Retha de La Harpe
Co-promoter: Dr Louise Leenen

Cape Town
November 2018

DECLARATION

I, Frans Hendrik Botes, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology or any of the participating parties.

Signed **Date**

ABSTRACT

With the constant evolution of information systems, companies have to acclimatise to the vast increase of data flowing through their networks. Business processes rely heavily on information technology and operate within a framework of little to no space for interruptions. Cyber attacks aimed at interrupting business operations, false intrusion detections and leaked information burden companies with large monetary and reputational costs. Intrusion detection systems analyse network traffic to identify suspicious patterns that intent to compromise the system. Classifiers (algorithms) are used to classify the data within different categories e.g. malicious or normal network traffic. Recent surveys within intrusion detection highlight the need for improved detection techniques and warrant further experimentation for improvement. This experimental research project focuses on implementing swarm intelligence techniques within the intrusion detection domain.

The Ant Tree Miner algorithm induces decision trees by using ant colony optimisation techniques. The Ant Tree Miner poses high accuracy with efficient results. However, limited research has been performed on this classifier in other domains such as intrusion detection. The research provides the intrusion detection domain with a new algorithm that improves upon results of decision trees and ant colony optimisation techniques when applied to the domain. The research has led to valuable insights into the Ant Tree Miner classifier within a previously unknown domain and created an intrusion detection benchmark for future researchers.

Keywords: *Ant Tree Miner, intrusion detection, decision trees, machine learning, swarm intelligence*

ACKNOWLEDGEMENTS

I wish to thank:

- Prof Retha De La Harpe for her assistance and guidance throughout the entire research project. Her assistance with last-minute funding for a trip in January 2017 and her guidance on the fundamentals of research ensured the success of the project
- Dr Louise Leenen for analysing every detail and providing an honest and realistic outlook on every task. Thank you for taking the time for numerous coffee meetings, which ended up in several hours of discussion. Thank you for numerous arrangements that had to be made to ensure the CSIR trip was a success

The financial assistance of the University Research Fund towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the University Research Fund.

The assistance of the Council for Scientific and Industrial Research, especially the Cybersecurity Centre for Innovation towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the Council for Scientific and Industrial Research.

DEDICATION

This thesis is dedicated to my parents, Lukas and Jacolise Botes. To the memory of my mother, although she was my inspiration to pursue my Master's degree, she was unable to see it completed. This is for her.

Then, special thanks to my father.
Thank you.

Thank you for the advice, your sacrifice and motivation throughout all the years. Thank you for buying and shipping a nagging nine-year-old boy a copy of Visual Basic 6 for his birthday to the remote Karoo. Thank you for tolerating the dissection of every electronic device throughout the house. Thank you for the library days and digitising my life from an early age. I would not have been able to do this without your relentless support. Thank you for everything, but most of all thank you for setting the example. For teaching me to grind it out, to have grit and to always give 200%.
This is for you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PUBLICATIONS

1. Botes, F., Leenen, L. & De La Harpe, R. 2017. Ant colony induced decision trees for intrusion detection. M. Scnalon & N.-A. Le-Khac (eds.), *Proceedings of the 16th European Conference on Cyber Warfare and Security, Dublin, 29 June 2017*. Reading, United Kingdom: Academic Conferences and Publishing International: 53-62.
2. Botes, F., Leenen, L. & De La Harpe, R. 2017. Ant Tree Miner Amyntas: automatic, cost-based feature selection for intrusion detection. *Journal of Information Warfare*, 73-92.

LISTINGS

GLOSSARY

ACDF Ant Colony Decision Forest

ACDT Ant Colony Decision Tree

ACK Acknowledge

ACO Ant Colony Optimisation

ACS Ant Colony System

ATM Ant Tree Miner

CART Classification and Regression Trees

DARPA Defence Advanced Research Project Agency, US

DDoS Distributed Denial of Service

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

DoS Denial of Service

ELC Experiment Life Cycle

FTP File Transfer Protocol

HIDS Host Intrusion Detection System

ICMP Internet Control Message Protocol

IDES Intrusion Detection Expert System

IDS Intrusion Detection System

IP Internet Protocol

KDD Knowledge Discovery and Data Mining

nACDT Novel Numerical Ant Colony Decision Tree

NIDS Network Intrusion Detection System

POD Ping of Death

R2L Remote to Local

ROC Receiver Operating Characteristic

SMOTE Synthetic Minority Oversampling Technique

SSH Secure Shell

SYN Synchronise

TCP Transmission Control Protocol

U2R User to Root

UDP User Datagram Protocol

VM Virtual Machine

VOIP Voice over Internet Protocol

YAGA Yet Another Get Admin

CHAPTER 1

INTRODUCTION

The chapter outlines an introduction to the research. Section 1.1 introduces the challenge and motivation for the work presented in the thesis. The problem statement is provided in section 1.2. The hypothesis and research questions undertook are outlined in sections 1.2 and 1.3 respectively. The core outcome of the research is discussed in section 1.5, followed by the research methodology in section 1.6.

1.1 Background

With the constant evolution of information systems, companies have to acclimatise to the vast increase of data flowing through their networks. News reports bombarded with stories of information breaches and hackers claiming ransom for company data have become the norm the past few years (Edwards, 2016; Heater, 2016). It is clear we live in a world where data requirements have become dynamic, and things have changed. The field of intrusion detection, however, has not. Traditional detection methods are still favoured for commercial products promoting a rigid, manual and static detection platform (Kumar, 2007; Hoque, Mukit, Bikas & Sazzadul Hoque, 2012; Aghdam & Kabiri, 2016). An Intrusion Detection System (IDS) analyses network traffic to identify suspicious patterns, with the intention to compromise the system. We train classifiers (algorithms) to classify the data within different categories e.g. malicious or normal network traffic. The improvement of intrusion detection classifiers is an ongoing research area as new attacks and improved algorithms are discovered almost daily (Agathou & Tzouramanis, 2008; Bilge & Dumitras, 2012; Aggarwal & Sharma, 2015; Burdette, 2016).

Despite research performed within the intelligent intrusion detection domains, little to none of the research projects have found a very effective solution (Lee, Fan, Miller, Stolfo & Zadok, 2002; Sommer & Paxson, 2010; Mohammad, Sulaiman & Khalaf, 2011; Burdette, 2016). They have, however, proved that it is possible to create intelligent intrusion detection components. There is, however, a constant drive to find improvements. Research has demonstrated that decision trees and random forests are among the best classifiers to use within the intrusion detection domain (Albayati & Issac, 2015).

Ant colony optimisation although unconventional within intrusion detection, has significant value when used to solve optimisation problems or even classify data (López-Ibáñez, Stützle & Dorigo, 2017). It is clear that machine learning, the science of giving computers the competence to adapt and learn without being explicitly programmed, could have great potential when applied within intrusion detection domains. This experimental research project focuses on the recent advances in machine learning, specifically the Ant Tree Miner (ATM).

The ATM classifier proposed by Otero, Freitas & Johnson (2012) allows for ant colony optimisation to induce decision trees. The proposed classifier builds decision trees using ant colony optimisation instead of using traditional C4.5 or CART techniques. This hybrid classifier showed high accuracy and excellent effectiveness during initial experiments, improving upon traditional decision tree classifiers. Decision trees, random forests and ant colony optimisation have been a very popular research topic when applied to the intrusion detection domain.

The research improves upon flaws within previous research by extensively ensuring reliability, comparability and reproducibility throughout the experimentation process (Sommer & Paxson, 2010; Papernot, McDaniel, Sinha & Wellman, 2016). To the best of the researchers' knowledge, the classifier combining decision trees with ant colony optimisation has not been tested within the intrusion detection domain. For further information, refer to the summarised timeline 5.1 in Chapter 5. The researcher therefore proposes, based on the evidence provided by previous research, to experiment with the Ant Tree Miner within the intrusion detection domain (Buczak & Guven, 2016; Folino & Sabatino, 2016; Kevric, Jukic & Subasi, 2016; Sahasrabuddhe, Naikade, Ramaswamy, Sadliwala & Futane, 2017).

1.2 Problem statement

As the influx of mischievous and malevolent activities becomes more advanced and adaptable, intrusion detection techniques are required to perform more intelligently to overcome more *avant-garde* attacks. Business processes rely on information technology and operate within a framework with little to no space for interruptions (Wange, Sahu & Mishra, 2016; Burdette, 2016). Cyber-attacks are aimed at interrupting businesses; false intrusion detections and leaked information burden companies with substantial costs, not just monetary but reputational as well (Shackelford, 2016).

Despite vast amounts of research conducted within the intrusion detection domain, little to none of the intelligent techniques used performed sufficiently reliable, efficient and accurate for commercial inclusion (Lee *et al.*, 2002; Sommer & Paxson, 2010; Mohammad *et al.*, 2011; Sahasrabuddhe *et al.*, 2017). Recent surveys within intrusion detection recommend improved detection techniques and warrant further experimentation for improvement (Amudhavel, Brindha, Anantharaj, Karthikeyan, Bhuvaneswari, Vasanthi, Nivetha & Vinodha, 2016; Chahal & Kaur, 2016; Wange *et al.*, 2016).

The Ant Tree Miner, a recent advance in the machine learning domain, can be considered a worthy candidate for further experimentation. The ATM combines the functionality of ant colony optimisation and decision trees. The latter is very popular with the intrusion detection domain (Albayati & Issac, 2015). Despite showing promising results by Otero *et al.* (2012), the algorithm and similar variants have had very little application within other domains. A literature review (table 5.1 in Chapter 5) reveals only three such applications, namely Boryczka, Probierz & Kozak (2016) in terms of categorising e-mails, Surjandari, Dhini, Rachman & Novita (2015) in terms of estimating the duration of dry docking, and Bursa & Lhotska (2015) with their ACO_DTree for biomedical data, as examples.

The research gap is highlighted with the research problem timeline (table 1.1), to summarise research performed in both the ATM and intrusion detection domains.

Table 1.1: Research problem timeline

2010	Sommer & Paxson (2010) note that despite extensive academic research, a striking gap exists in terms of the deployment of machine learning techniques within the intrusion detection domain.
2010	Tavallaee, Stakhanova & Ghorbani (2010) provide further insights into the problem as they note concerns about the validity of the techniques and procedures undertaken by researchers to solve the problem.
2012	Otero *et al.* (2012) create the ATM algorithm that induces decision trees using ant colony optimisation.
2014	The ATM is extended with the bagging ensemble technique by Chennupati (2014) as they raise concerns about the ATM's stability.
2015	Surjandari *et al.* (2015) extent a similar classifier called Ant Colony Decision Tree (ACDT) for estimating the duration of dry-docking.
2015	Bursa & Lhotska (2015) build the ant-inspired method called ACO_DTree over biomedical data.
2016	Boryczka *et al.* (2016) show the usefulness of ACDT to classify e-mails.
2016	Wange *et al.* (2016) highlight that ant colony optimisation should be combined with other machine learning techniques due to its threshold determination abilities when applied within intrusion detection.
2016	Chahal & Kaur (2016) note that misuse detection is not adequate to solve the current intrusion detection problem and further development of anomaly detection techniques is required.
2017	Rudd, Rozsa, Gunther & Boult (2017) outline the several flawed assumptions inherent to machine learning algorithms that prevent mapping stealth malware.

1.3 Hypothesis

A machine learning task undertakes the concept of a hypothesis model (Domingos, 2012). We theorise that a hypothesis is based on statements made about reality and known facts (Goodman, 1999), thus, the hypothesis is a particular framework, at a given point in time, describing the correlation between variables. In machine learning a hypothesis is developed for each machine learning model as we consider input variables (training data set) and output variables (classification model) to predict a given outcome (Agarwal & Dhar, 2014). A test of the hypothesis would be against the test data set from which evaluation metrics such as a confusion matrix is derived (Agarwal & Dhar, 2014). We further discuss the role of the confusion matrix in testing the hypothesis for each model in section 6.5.1.

The hypothesis for this research project is that the ATM classifier can be used to classify intrusion data and compared in terms of accuracy with traditional decision tree classifiers when applied within the intrusion detection domain.

1.4 Research questions

Each chapter of the thesis undertakes to address the stated hypothesis where a specific research question will be answered to obtain insights into a particular topic related to the hypothesis. The research questions pose a platform for the development of the experiments. It is therefore necessary to consider each question not only individually, but also as part of a holistic contribution towards the outcomes. The research questions considered in the thesis are as follows:

Table 1.2: Research questions

Question	Description	Thesis relation
1	What are the current pitfalls of implementing machine learning techniques within the intrusion detection domain?	Chapter 2 Chapter 3 Chapter 6
2	How do decision trees and ant colony optimisation techniques perform when applied within the intrusion detection domain?	Chapter 4 Chapter 6
3	How should intrusion detection systems be evaluated?	Chapter 6 Chapter 7
4	How does the Ant Tree Miner classifier perform when applied within the intrusion detection domain?	Chapter 6 Chapter 7

1.5 Outcome

The research undertaken covers a broad spectrum, from intrusion detection and machine learning to cyber security. The main focus of the research is to establish the ATM classifier for data classification of intrusion detection data that improves, in accuracy, on the previously used methods, thereby leading to suggesting recommendations on improving the performance of the classifier when applied to intrusion detection data. However, the research makes contributions towards general knowledge within other domains as well. Additionally, the research derives a benchmark for ATM or variant algorithms within intrusion detection to be used by future researchers.

1.6 Research methodology

The research is aimed at the computer science community with focus areas in machine learning and cyber security. Deductive reasoning has been applied throughout. The primary goal of the study was to create and contribute theory with regard to the ATM classifier within the intrusion detection as well as machine learning domains. The strategic approach to the research study has been experimental. The researcher experimented with algorithms using the NSL-KDD data set. The data set created by Tavallaee, Bagheri, Lu & Ghorbani (2009) improves upon the issues within the famous KDD'99 data set, used in several research projects. The research analyses both data sets in detail in Chapter 6. The research mostly used quantitative analysis and the choice for analysis could be considered as a mono method. The results generated have been analysed statistically using the performance derivatives, Receiver Operating Characteristic (ROC) analysis and cost evaluation. Cross-sectional research design was adopted, as quantitative analysis measures have been used to measure

the performance and accuracy of the classifier on intrusion detection data sets at a given period. We can draw similarities between hypothesis testing and the traditional confusion matrix used to analyse machine learning problems. It is common for a hypothesis to be derived for each machine learning task as explained during the hypothesis statement of the research of Agarwal & Dhar (2014). The research methodology is further discussed in Chapter 6.

1.7 Structure

The thesis is organised as follows: Chapter 2 provides a background and introduction to intrusion detection systems, attack classification and the detection methodologies. This is followed by a deeper review of machine learning, specifically within the intrusion detection domain, in Chapter 3. The problems and challenges described in the previous chapter form the focus for the intelligent classifiers introduced in Chapter 4. As the deficiencies of these intelligent classifiers are revealed, we look into the proposed ATM classifier for improvement in Chapter 5. Chapter 6 details the experiment to establish the classifier within the intrusion detection domain. In the final chapters, we analyse and summarise the results in Chapter 7 and outline research challenges with suggestions for further improvement in Chapter 8.

CHAPTER 2

INTRUSION DETECTION SYSTEMS

Chapter 2 introduces the intrusion detection domain in section 2.1, and we dig deeper by looking into the different topologies in section 2.2. Attack classification, specifically related to the data set, is discussed on a very high level in section 2.3, and the detection thereof is complimented in section 2.4. We place the current state and detection techniques under scrutiny as we look into the limitations of intrusion detection systems in section 2.5.

2.1 Introduction to intrusion detection systems

Although locks, immobilisers and other security features secure most vehicles, cars are still at risk of being stolen. We can consider a scenario where a stranger walks up to a car, inspects the car, looks around and then moves towards the front door, trying to open it. The door is locked. The stranger moves closer towards the window and gently tries to pry it open. The window too is closed and leaves the stranger's attempts unsuccessful. The vehicle is clearly secured, so why install an alarm? The need for implementing an IDS (alarm) is often questioned by many security architects (Khan, Gani, Wahab, Shiraz & Ahmad, 2016). Why bother? With firewalls, patched operating systems and sound password controls within computer environments should be secure. The answer is simple, as the facts state that with all the controls in place, cars still get stolen. Computer environments still get penetrated, and the amount of data harboured and transported throughout company networks has vastly increased in recent years (Morrow, 2012).

The data boom has left companies with significant vulnerabilities waiting to be exploited. Malicious attacks have grown more sophisticated and widespread, even targeting smaller companies with hopes to exploit security weaknesses (Paulsen, 2016; Kent, Tanner & Kabanda, 2016). Intelligent systems are one of the most innovative technologies currently used within the information security field, with high potential to narrow the wide gap between the lines of malicious activities and security control activities (Maitra & Madan, 2017; Harel, Gal & Elovici, 2017).

Small, medium and large organisations have identified the need to implement information security solutions and governance activities moving towards controls and principles protecting information between processes, technology and people (von Solms & Van Niekerk, 2013). Most effective and core essential solutions to information security include firewalls, access controls and encryption (Wu & Banzhaf, 2010). These can all be considered as traditional intrusion prevention techniques that are failing daily to protect organisations from the increasingly sophisticated pernicious activities.

Bilge & Dumitras (2012) performed an empirical study on zero-day attacks; in the real world, typical zero-day attacks last 312 days on average. The problem worsens, as Burdette (2016) notes that most successful corporate attacks only get discovered on average after 314 days. The statement is significant as attackers could remain within a network for extended periods without being detected. Zero-day attacks refer to vulnerabilities not previously disclosed to the general public that can be exploited. Once the vulnerability is disclosed publicly, the number of exploits increases by a factor of five (The Economist Intelligence Unit, 2016). This alarming statement is further supported by Frei (2014) from FireEye, a well-known cyber security research company, stating that on average, attacks remain undetected for at least 312 days. It is evident that the real crux within the cyber security field lies with detection. It is difficult to prevent something you are unaware of, therefore without any detection, no controls will be able to satisfy the underlying risk properly.

Kemmerer & Vigna (2002) define intrusion detection as the process of distinguishing between malicious and normal network activities. The core structure and component set for any IDS consists of the following:

- Sensors or agents – gather information
- Data sources and data repository – store information
- Detection algorithm – analyses information
- Response system – responds to analysis
- Management system – monitors, analyses and configures IDS

Initial intrusion detection was performed by system administrators monitoring user activities on their computers. The late 1970s and early 1980s introduced an approach for intrusion detection based on audit logs. The oldest system topology to have been used in intrusion detection is Host Intrusion Detection System (HIDS), discussed in section 2.2.2 (Debar, Dacier & Wespi, 1999).

With limited external network connections, system administrators would manually monitor logs to determine if an intrusion occurred. The manual method of intrusion detection was done on an *ad hoc* basis and was not scalable at all (Kemmerer & Vigna, 2002). As storage and logging capabilities grew, intrusion detection significantly adopted the approach. Detection now relied on analysing the data received to identify intrusive behaviour. Dorothy Denning and Peter Neuman proposed such systems based on an intrusion detection model and most research focus on constructing accurate and effective detection models (Denning, 1987; O'Leary, 1992; Kemmerer & Vigna, 2002; Yost, 2016). As the attacks grew more sophisticated, in the early 1990s intrusion detection systems evolved to a combination of expert systems using statistical approaches as the core model of detection (Kemmerer & Vigna, 2002; O'Leary, 1992).

Statistical expert systems offered real-time detection capabilities, contributing to the continued growth in knowledge and capability for intrusion detection. The models later shifted towards more automated events that gathered knowledge on normal and abnormal patterns; anomaly

detection is discussed further in section 2.4.2 of this chapter. These 'modern' systems compared the behaviour of different profiles to identify malicious activities (Patcha & Park, 2007). The systems could easily be crippled by data designed to mislead the system or changes in the processes being measured for comparisons. The system would create a filter based on the user's behaviour pattern, and attackers could spoof the filters by imitating the behaviour. Hackers could also camouflage their malicious activities by establishing high variance behaviour within the system. The large set of intrusion detection data profiling the user's patterns was non-stationary and had to be updated each time the user's behaviour changed. This required system designers to purge data whenever the process changed (Kemmerer & Vigna, 2002; O'Leary, 1992).

The challenges experienced with intrusion detection systems have led to the creation of multiple intrusion detection topologies as the creators experimented with different setups and configurations.

2.2 Intrusion detection topologies

IDS technologies are classified into three categories, namely network-based, wireless and host-based technologies, each depending on the type of events monitored by the system and the location of the IDS within the network environment (Scarfone & Mell, 2007). For this thesis, only the popular network and host-based topologies are discussed.

2.2.1 Network intrusion detection systems

A Network Intrusion Detection System (NIDS), as the name suggests, examines and monitors the network traffic and is therefore considered to be platform independent. Being implemented within the network layer, the NIDS is able to monitor traffic from a vast amount of hosts simultaneously. The NIDS uses anomaly and signature (rule-based) detection methods, explained in more detail in section 2.4 (Bhuyan, Bhattacharyya & Kalita, 2014).

There are several ways to implement NIDSs, most commonly with port mirroring on a configured and compatible switch, network taps or hub connections (Bushev, Vlasenko, Glotov, Monakhov & Tishin, 2013). The most favoured implementation is between the firewall and the various hosts in the network (Hamid, Sugumaran & Balasaraswathi, 2016). Figure 2.1 illustrates the network layout where NIDSs can be used. The NIDS mentioned in figure 2.1 uses a network tap to connect to the network. The NIDS captures information from the network and analyses each stream of packets for malicious activity or deviation from the normal network traffic. For each network packet, the NIDS monitors the IP and transport layer headers to filter malicious content.

Network-based attacks such as Denial of Service (DoS) attacks, probe attacks and malicious monitoring are just some of the examples where a NIDS would be more effective in detecting the attacks than a host-based system (Lin, 2013). Depending on the detection platform, the ability to build profiles based on the observed behaviour and then compare it to the current behaviour adds an extra layer to the security profile.

Notwithstanding all the benefits, NIDSs do have some clear drawbacks that need to be mentioned. Although the NIDS can monitor several hosts and the traffic between them, the NIDS has very limited visibility inside these hosts as it is only implemented on the network layer. If the network traffic is encrypted, the NIDS cannot decrypt the traffic effectively in real-time. However, it is to be expected that the traffic behind the firewall is not encrypted and only once it leaves the firewall, becomes encrypted (Holden, 2004; Kizza, 2017). Liao, Richard Lin, Lin & Tung (2013) criticise NIDSs by mentioning that when anomaly detection is used, it tends to show high false positives and false negative rates, and is unable to provide full analysis when under extremely high loads. Figure 2.1, NIDS topology, outlines the traditional NIDS intrusion detection layout.

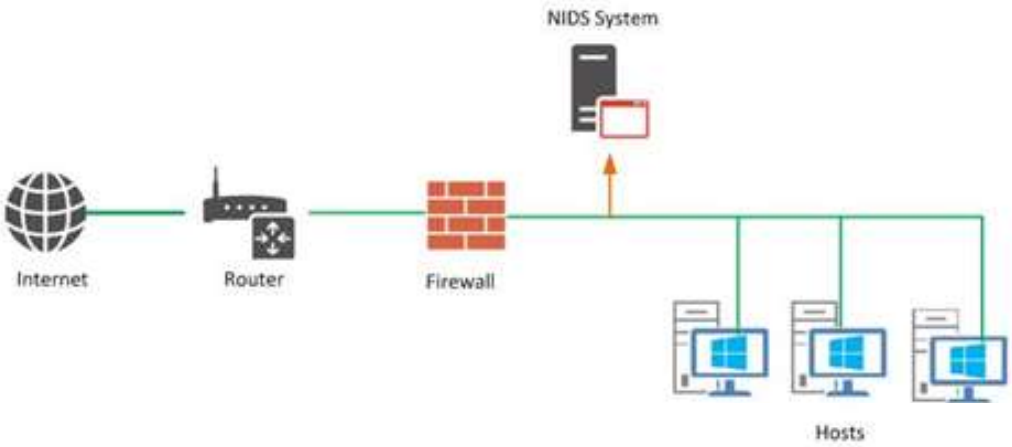

Figure 2.1: NIDS topology

2.2.2 Host intrusion detection systems

HIDSs, as the name suggests, examine and monitor the specific host on which it has been deployed (Vigna & Kruegel, 2005). The HIDS consists of a server and several agents, usually software-based on each host machine.

Unlike NIDSs, this system is platform dependent for the agents. The server can be placed anywhere as long as it is routing the traffic from the server and the agents are uninterrupted. The HIDS examines the host in more detail, looking at file system modifications, system calls and application logs, to mention a few (Liao *et al.*, 2013).

The HIDS builds profiles based on the host kernel and file system behaviours and then compares the current activities; upon deviation the system reports an attack. The accuracy and efficiency of HIDSs largely depend on the system characteristics monitored on the host. A large portion of the HIDS's success sits on the shoulder of the security administrator. Figure 2.2 illustrates an environment where agents are installed onto each host routing towards the server.

HIDSs are considered the flavour of the month within the cloud computing environment (Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013). A Virtual Machine (VM) or hypervisor allows

for much wider intrusion detection capabilities, based on its scalability and ease of implementation (Modi *et al.*, 2013). Best practice, however, would be to implement the HIDS behind the firewall in the network environment (McHugh, Christie & Allen, 2000; Verwoerd & Hunt, 2002; Hamid *et al.*, 2016). The HIDS also uses a combination of anomaly- and signature-based detection methods. Figure 2.2, HIDS topology, outlines the traditional HIDS intrusion detection layout.

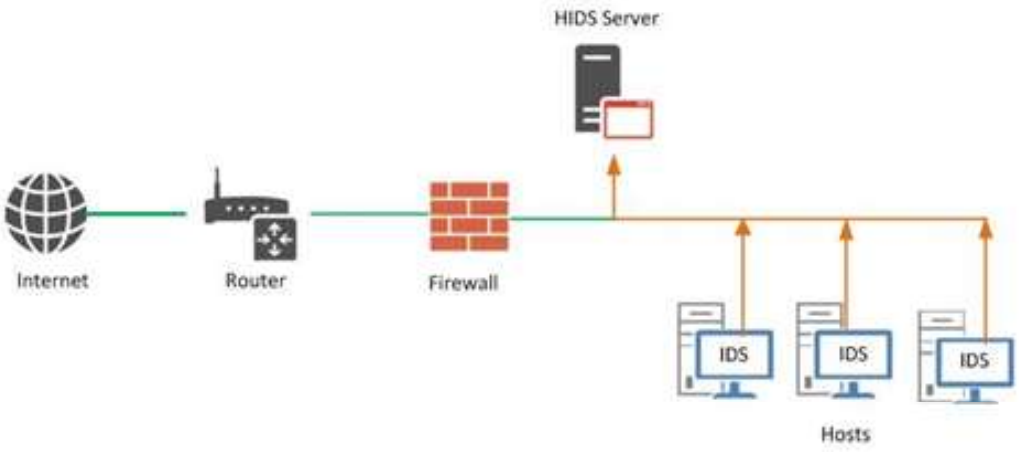

Figure 2.2: HIDS topology

Some HIDSs have extremely low resource consumption, further strengthening their cause within the cloud environment. As with NIDSs, HIDSs also have issues. Due to the system relying heavily on agents to perform the data collection, delays in alert generation and centralised reporting are common problems (Bukac, Tucek & Deutsch, 2012). In systems with limited resources, modern HIDS agents can cripple the system or contribute to an unstable host environment (Bukac *et al.*, 2012). Its inability to look "beyond" the current host is overcome when using intelligent management components (Bukac *et al.*, 2012).

Host-based systems rely heavily on the logs produced from a system and therefore require a host to be configured correctly to produce the logs. It is a known fact that logging can sometimes increase resource consumption (Ristic, 2005). Extensive logging leads to the HIDS-based system being less cost effective; due to specific configurations being required, the environment will have to be reconfigured or sometimes even be upgraded. The network-based system has a much lower cost entry point, being easier to install and requiring fewer configurations to be done; it does however not provide the accuracy that normal host-based systems would (Kelly, 2006; Lin, Zhang & Ou, 2010). Kelly (2006) compared the systems and favours NIDSs for the following reasons:

1. **Ownership costs**: Low costs for wide coverage; single systems can detect intrusion directed across the entire network of hosts.
2. **Analysis**: Perform packet analysis on network traffic containing protocol fields and payload data.
3. **Early detection**: Ability to capture attacks that do not succeed due to monitoring the network before it reaches hosts.

4. **OS independence**: The system works regardless of the operating systems used, allowing for easier implementation.

The added advantage of OS independence can be heavily debated. Research has already shown that NIDSs do not operate with the same accuracy (Lin *et al.*, 2010; Kelly, 2006). It can be noted that it is partly due to system independence, without the added knowledge that the system does not discriminate against data from different systems. Being able to do so might increase the accuracy as seen with HIDSs.

In some cases, network and host-based systems are combined to form a hybrid system. One can argue that this approach will be more cost effective within an environment that already has a host-based system in place. Due to the costs and fewer configuration requirements, the option becomes much more viable. A company currently only using a network-based system might require more accurate results. The hybrid approach has a myriad of advantages, but the most attractive one is the ability to integrate (Peddabachigari, Abraham, Grosan & Thomas, 2007). The network-based system might be able to detect an attack and use the host-based system to verify if this attack is legitimate before it reports. This system will drastically decrease detection issues found in network-based systems (Peddabachigari *et al.*, 2007).

It is clear that these systems complement each another when used in conjunction with one another. With the hybrid approach, the central repository will receive information from both topologies, allowing for a wider net to build detection signatures. The hybrid system allows the IDS managing component to push specific signatures to agents on an *as-needed* basis due to the network layer still being covered. The hybrid approach operates more efficiently and effectively, as it overcomes drawbacks in both systems when implemented separately (Chauhan & Chandra, 2013). Kelly (2006) explains that the different characteristics should be embraced by combining both network and host systems for increased performance and accuracy.

2.3 Attack classification

Similar to police profiling a criminal in an active investigation, we can profile cyber attacks as well. It is therefore crucial to understand each attack. Kroll, Barocas, Felten, Reidenberg, Robinson & Yu (2016) identifies three characteristics to which a system must comply in order not to be considered compromised or attacked:

- System/user actions must conform to statistically predictable patterns
- System/user actions must not portray sequences of commands to subvert security policies
- Actions of processes must comply with the described specifications for allowable actions

If any of the three characteristics are not met, the system could be considered under attack or compromised. Detected attacks can be classified into four different categories, namely User to Root (U2R), Remote to Local (R2L), DoS and probe attacks, further discussed below (Engen, 2010). We look at the attacks on an introductory basis and by no means cover all the different variations. The researcher's focus is placed on the attacks within the data sets used for this

research project and the detection thereof.

Although some attacks within the data set are considered old, comparison to recent attacks are drawn in the literature study. The outline should be read as an introduction and is by no means a comprehensive updated list of attacks. For further information on the data set, refer to Chapter 6. A comparison of data sets are provided in table D.1. The comparison outlines the popularity of the NSL-KDD data set within the research communities. During the time of experimentation (2016), the NSL-KDD data set provided comparable research results. The methodology discussion in section 6.8 outlines the research methodology required to compare results between research studies.

2.3.1 User to Root (U2R)

The U2R attack focuses on the attacker trying to gain administrative (root) access to the system by exploiting vulnerabilities. The vulnerabilities would mostly be within the operating system or underlying software used on the system. The attacker would therefore first have to be able to access the system in a normal account environment and then try to gain root access (Bhattacharyya & Kalita, 2013). We discuss R2L where the attack would obtain local access in section 2.3.2.

U2R attacks could be arduous to distinguish from normal network traffic. Some examples of these attacks include buffer-overflow attacks, loadmodule, Perl and rootkits. Ahmad, Abdullah & Alghamdi (2010) highlight how U2R attacks are the most dangerous of all and can cause severe losses for companies. It is no surprise that with administrative abilities on any system, the user has no limitations on the system. The attack platform can spread from Android, web browser plugins, software and operating systems, to name a few. Although widespread, exploiting U2R vulnerabilities is difficult to achieve as it requires the attacker to already have access to the system and the skill set to perform such advanced attacks.

To summarise the characteristics and features of U2R attacks, table 2.1 is adopted from Kendall (1999) who created the DARPA data set, widely known for intrusion detection. The DARPA data set was adopted to become the KDD Cup '99 (KDD'99) data set (KDD Cup, 2016). The researcher further updated the table to include more modern U2R attacks in the NSL-KDD data set, used within this research project (Tavallaee *et al.*, 2009). For further information on the NSL-KDD data set, refer to Chapter 6.

Kendall (1999) elects the buffer overflow attack as one of the most popular U2R attacks. The statement is based on the findings of Anderson (1972) who already detected the attack 44 years ago. Buffer overflow attacks can be performed whenever the vulnerability exists, creating a vast threat landscape. Buffer overflow attacks form a substantial portion of all security attacks due to its simplicity and typical vulnerability (One, 1996; Cowan, Wagle, Pu, Beattie & Walpole, 2003; Black & Bojanova, 2016). Due to the popularity and widespread usage, we will further discuss the attack below. Buffer overflow attacks can occur in several instances, summarised as follows (Nelissen, 2004):

- Input into graphical user interface

- Data sent over network to program
- Data within a file
- Data within command line

Table 2.1: U2R attacks summarised adapted from Kendall (1999) & Tavallaee et al. (2009)

Name	Service	Vulnerable platform	Mechanism	Time required	Impact
Eject	Any (telnet, rlogin)	Solaris	Exploit bug (buffer overflow)	Medium	Root shell
Ffbconfig	Any (telnet, rlogin)	Solaris	Exploit bug (buffer overflow)	Medium	Root shell
Fdformat	Any (telnet, rlogin)	Solaris	Exploit bug (buffer overflow)	Medium	Root shell
Loadmodule	Any (telnet, rlogin)	SunOS	Exploit bug	short	Root shell
Perl	Any (telnet, rlogin)	Linux	Exploit bug	short	Root shell
Ps	Any (telnet, rlogin)	Solaris	Exploit bug	short	Root shell
Xterm	Any (telnet, rlogin)	Linux	Exploit bug (buffer overflow)	short	Root shell
Anypw	Any (telnet, rlogin)	NT	Exploit bug	short	Root shell
Casesen	Any (telnet, rlogin)	NT	Exploit bug	short	Root shell
Ntfsdos	N/A manual	NT	Feature abuse	short	Root file access
Sechole	Any (telnet, rlogin)	NT	Feature abuse	short	Root shell
Yaga	Any (telnet, rlogin)	NT	Feature abuse	short	Root shell

Kendall (1999) construes that buffer overflow attacks try to copy too much data into a static buffer, thereby exploiting the paucity of controls validating the data. To explain the attack vector, the diagram below is used, illustrating how buffer B is pushed beyond the scope intended.

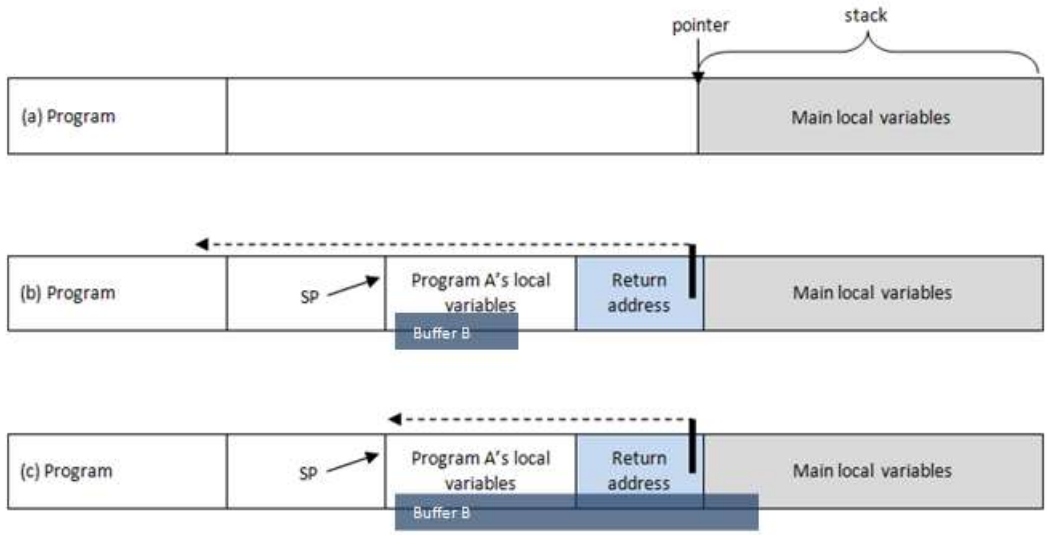

Figure 2.3: Buffer overflow attack

Normal operating conditions are observed within (a), as the program has not initiated any command yet. **Command A** is called in (b) and initiates the input hosted within buffer B. Due to the buffer overflow (c), the address space for B has now increased significantly within the

address space, allowing the attacker access to virtual space not previously usable. Stack smashing as indicated in figure 2.3 is one of the simplest buffer overflow attacks available (Larochelle & Evans, 2001). The attack will overwrite a buffer on the stack to replace the return address. Instead of jumping back to the return address, the control will jump to the address placed on the stack. Once within the address space, the attacker can execute arbitrary commands.

To simplify, let us assume a program requires the user to input their last name. The programmer would have to decide how many characters that field buffer requires e.g. 20 characters. If the user inputs a value of 25 characters, the last five would overflow the last name buffer. When this transpires, the residue characters will be placed onto the program stack, overwriting the subsequent commands that should be executed. By manipulating the overflow data in the stack, the attacker can cause arbitrary commands to execute within the operating system.

These attacks pose a high risk as an attacker can execute any command; however, insights into the stack and exploitation thereof are extremely technical and not easy to execute (Black & Bojanova, 2016). Technical countermeasures help diminish the threats; it is, however, important to note that most buffer overflow vulnerabilities exist due to negligence or errors by programmers. Black & Bojanova (2016) note that in specific instances, buffer overflow attacks are exceptionally hard to defend against. The attack is worthy of discussion as it constitutes a vast amount of security issues and does have a substantial impact when exploited.

Another attack, Yet Another Get Admin (YAGA), involves exploiting the user's registry and is classical to the U2R nature. The attacker requires local access and couples with other attack methods to fully exploit the victim's machine. Korba (2000) describes the process of exploitation as follows:

1. Obtain local access to machine.
2. Upload attack files (telnet).
3. Setup registry files.
4. Crash service.
5. User added to domain admin group.

The attack abuses the debugger program in Windows by changing the registry to launch the attack commands instead of the default debugger. Whenever the change is completed, the attacker will launch another attack to crash services on the computer. Once completed, the user will be added to the Domain Admins group.

It is important to note that with only two of the mentioned U2R attacks, physical access to the device is required. We refer to these attacks as local-based attacks e.g. AnyPW and NTFSDOS (Cowan *et al.*, 2003). The other attacks can all be executed remotely, and whenever the attacker gains administrative access the system is severely jeopardised. However, to execute these attacks, local access is required before an attacker can escalate its

privileges to administrative. Based on this we can argue that it is important to mitigate local account exploitation as an additional security layer.

2.3.2 Remote to Local (R2L)

Very similar to U2R attacks, with R2L attacks the intruder does not have access to an account on the host and therefore needs to obtain local access to a network connection. R2L could be considered as a prequel to U2R attacks. The attacker would send packets to a remote machine or network host without being a legitimate user. Bhattacharyya & Kalita (2013) name buffer overflow and unverified input attacks as the two most commonly used R2L attacks.

We have already discussed buffer attacks in more detail during the U2R section. The attacker would, therefore, exploit misconfigurations in the security policies or engage in social engineering. With social engineering, the human operator is targeted and tricked instead of a system. Some practical examples would be to take advantage of common anonymous File Transfer Protocol (FTP) misconfigurations, guest accounts, etc.

To summarise the characteristics and features of R2L attacks, table 2.2 below is adopted from Kendall (1999) who created the DARPA data set, widely known for intrusion detection. The DARPA data set was adopted to become the KDD Cup '99 data set (KDD Cup, 2016). The researcher further updated the table to include more modern DoS attacks as included in the NSL-KDD data set, used within this research project (Tavallaee *et al.*, 2009). For further information on the NSL-KDD data set refer to Chapter 6.

Hoque, Bhuyan, Baishya, Bhattacharyya & Kalita (2014) iterate the difficulty of detecting R2L attacks due to network traffic resembling normal characteristics. The statement is based on Thomas, Sharma & Balakrishnan (2008) who emphasise the low variance within the attacks, hindering the ability to detect with unique signatures or anomaly detection. In another study, Tupakula, Varadharajan & Akku (2011) highlight the high amount of the network level and host level features within R2L attacks as the main cause for detection failure.

The R2L attack category offers a diverse set of attacks ranging from execution to implementation. Intrusion models currently proposed for R2L attacks fail to perform with desirable detection and low false alarm rates (Sabhnani & Serpen, 2003; Anwar, Mohamad Zain, Zolkipli, Inayat, Khan, Anthony & Chang, 2017). To illustrate the simplicity and severe implications, we discuss a normal dictionary attack and a netbus attack. Dictionary attacks are the most efficient and frequently used attacks before attempting brute-force attacks (Dey, 2016). The dictionary can be created by using generally used entries or more complicated entries based on information specifically targeting the user. Figure 2.4 below was created to visualise a very basic dictionary attack using general entries.
To specifically target the user, the attacker would have to engage in social engineering to gain more information about the user. The information gained will then be used to build a specific dictionary related to the target. Information such as birth dates, first names, middle names and even family or spouse names are commonly shared on social media and are heavily favoured by users when creating passwords. Figure 2.5 visualises the attacker building a dictionary based on the information gained from the victim's social media profile to illustrate a simple

Table 2.2: R2L attacks summarised adapted from Kendall (1999) & Tavallaee et al. (2009)

Name	Service	Vulnerable platform	Mechanism	Time required	Impact
Dictionary	telnet, rlogin, pop, imap, ftp	All	Feature abuse	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Httptunnel	http	All	Feature abuse	Long	Request information using cookies
Imap	imap	Linux	Exploit Bug	Short	Root Shell
Named	dns	Linux	Exploit Bug	Short	Root Shell
ncftp	ftp	All	Feature abuse	Medium	Execute commands
netbus	X, telnet	NT	Feature abuse	Short	Remote administration tool
netcat	X, telnet	NT	Feature abuse	Medium	Remote access
Phf	http	All	Exploit Bug	Short	Execute commands as user http
ppmacro	X	NT	Feature abuse	Long	View user files
Sendmail	smtp	Linux	Exploit Bug	Long	Execute commands as root
sshtrojan	SSH	Linux	Feature abuse	Long	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor keystrokes remotely

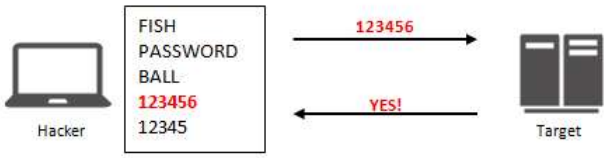

Figure 2.4: Basic dictionary attack

social engineering dictionary attack.

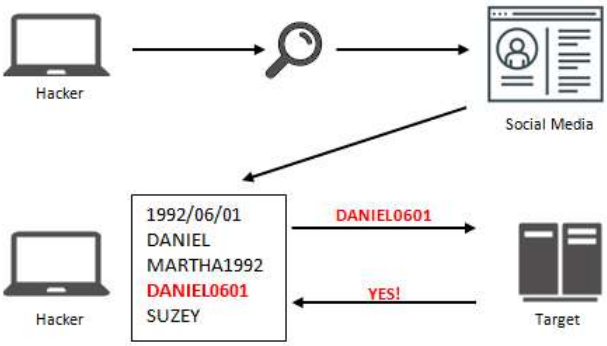

Figure 2.5: Dictionary attack with social engineering

One can imagine that it is almost impossible to detect these attacks, however mitigating

controls such as account locking, strong password policies, etc. are of great importance (Canetti, Halevi & Steiner, 2006). Using the same password for several sites almost instantly becomes a vulnerability. The hacker would have a broad range of tools available to assist with building dictionaries and gathering information for it. Unlike dictionary attacks, netbus attacks require more skill and patience. Netbus attacks require the attacker to gain the trust of the target. The attacker would stage a scenario that could trick the victim such as e.g. death of a relative, credit card fraud or the collection of prizes. The victim would have to open and download the malicious content. Usually, the trojan is distributed via e-mail to the user (Kulakow, 2001; Chen, Wei & Delis, 2008). The attacker would use his imagination here, and sometimes these attacks are heavily a hit and miss depending on how gullible the victim is. Classic netbus attacks are illustrated in figure 2.6.

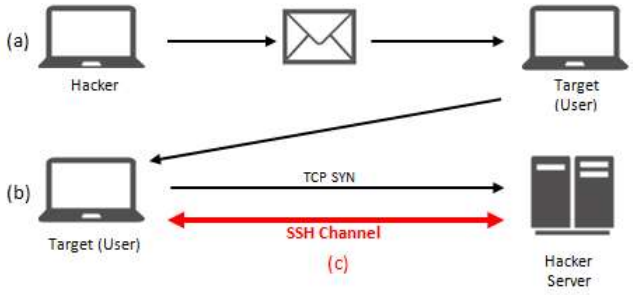

Figure 2.6: Classic netbus attack

Within (a) the hacker sends the user an email containing a trojan that opens a backdoor on the user's machine. A backdoor is designed to hide inside a target host, giving the attacker access to the system for later use. Once the user opens the email attachment that executes the backdoor (b), a Transmission Control Protocol (TCP) Synchronise (SYN) request is initiated to the hacker's server. The server would be waiting and listening for the requests from a specific port to complete the traditional TCP 3-way handshake. Once this is done, the hacker's server will establish a Secure Shell (SSH) tunnel on top of the TCP socket, illustrated with (c) (Stiawan, Idris, Abdullah, AlQurashi & Budiarto, 2016).

Once the SSH tunnel is created the hacker can unknowingly transfer arbitrary commands to the victim's computer. Not nearly as eluding as with dictionary attacks, netbus attacks can also be "tweaked" to become stealthier; by encrypting the commands within the tunnel any IDS can be evaded (Stewart, 2013). The R2L attack is much more dangerous than DoS or probe attacks (discussed below) due to the user obtaining direct access to the system. They do however sometimes require some user intervention as indicated in the attack scenario discussed, therefore social engineering can be used. Social engineering has previously been discussed as exploiting public information from a victim to perform an attack e.g. e-mail, facebook etc. (Baiardi, Corò, Tonelli & Sgandurra, 2014).

2.3.3 Denial of Service (DoS)

Speculated to have started in the late 1990s, DoS attacks are one of the most common attacks (Kessler, 2000; van Rijswijk-Deij, Sperotto & Pras, 2014). The purpose of DoS attacks is to interrupt and cripple services on a host. Simple yet highly effective, this attack has grown

in sophistication over the years. When combined with other attacks, the attacker can perform a multi-stage attack on the victim using the DoS to 'crash' or hinder services on the target host (van Rijswijk-Deij *et al.*, 2014).

Most known descriptions for these DoS attacks were made by Kendall (1999), who categorised it as follows: "abuse legitimate features", "take advantage of bugs in a particular daemon" and "create malformed packets that confuse the Internet Protocol (IP)/TCP stack of the machine trying to reconstruct the packet". The attack taxonomy has not changed much as several years later Hashmi, Saxena & Saini (2012) noted similar definitions for defining DoS attacks. Modern descriptions focus on Distributed Denial of Service (DDoS) attacks as outlined by Somal & Virk (2014), indicating an attack which uses a large number of computers to launch a coordinated DoS attack against a single machine or multiple victim machines.

The primary goal of DoS attacks is to constrain the available resources in such a way that the remaining resources become limited or unavailable to legitimate users. The attacker relies heavily on the computational costs to hinder services and even stop legitimate services from running successfully. When an attacker would like to disrupt a web server, the primary goal would be to bombard the host with a dummy request to cripple resources. These attacks could be categorised as the following types of DoS attacks (Rittinghouse & Hancock, 2003):

- **Flaw exploitation attacks**
  The attacker would try to exploit a vulnerability within the host to slow it down or exhaust the resources available. The popular Ping of Death (POD) involves the attacker sending a malicious ping to the host. In general, a ping would consist of 64 bytes or 84 when the IP header is included. Most computerised systems will not be able to handle pings larger than the maximum IP packet size of 65 535 bytes (Sharma & Kunwar, 2016).

  Sending such a ping, or larger, to the host computer would in most cases cause it to crash. Within intrusion detection systems, a signature that triggers whenever ICMP packets are longer than 64 000 bytes would potentially detect POD attacks.

  Some cases lead to certain vulnerabilities which will be exploited to implement DoS attacks such as Domain Name Service (DNS) amplification. In this specific attack, the Internet Control Message Protocol (ICMP) echoes messages to the target being bombarded. A signature could be devised to determine the ping of death. A study performed by van Rijswijk-Deij *et al.* (2014) note that DNSSEC used to mitigate DNS cache poisoning can be exploited to perform severe DoS attacks. This is a clear indication of how counteractive measures can sometimes increase the threat landscape.

- **Flooding attacks**
  This specific type of DoS attack has won flavour of the month for several years now (Wueest, 2014). The actual benefit lies in the simplicity, and the attacker sends more requests to the target host than it can manage. The host will then exhaust all processing capability or the network bandwidth. Currently, these DoS attacks are some of the hardest to combat as there is no vulnerability in the system being exploited; in

retrospect, any secure system can be targeted and fall victim. The attack can also be distributed and then becomes significantly more dangerous.

DDoS attacks use a resource pool to attack the victim host. The botmaster (hacker) can use any system exploited to form part of the attack. These attacks could be compared to firing a warning shot and are usually followed by more sophisticated attacks (van Rijswijk-Deij *et al.*, 2014).

To summarise the characteristics and features of DoS attacks, table 2.3 is adopted from Kendall (1999) as his DARPA data set is the origin of the data set used within this research. Modern DoS attacks in the NSL-KDD data set have been added to the table (Tavallaee *et al.*, 2009). For further information on the NSL-KDD data set refer to Chapter 6.

Table 2.3: DoS attacks summarised adapted from Kendall (1999) & Tavallaee et al. (2009)

Name	Service	Vulnerable platform	Mechanism	Time required	Impact
Apache2	http	Any Apache	Feature abuse	Short	Crash httpd
Back	http	Any Apache	Feature abuse / Exploit Bug	Short	Slow server response
Land	N/A	SunOS	Exploit Bug	Short	Freeze machine
Mailbomb	smtp	All	Feature abuse	Short	Annoyance
SYN Flood	Any TCP	All	Feature abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None	Exploit Bug	Short	Possible crash, freezing
Process Table	Any TCP	All	Feature abuse	Moderate	Deny new processes
Smurf	icmp	All	Feature abuse	Moderate/ Long	Network Slowdown
Syslogd	syslog	Solaris	Exploit Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Exploit Bug	Short	Reboot machine
Udpstorm	echo/ chargen	All	Feature abuse	Short	Network Slowdown

Due to its simplicity, DoS attacks have significantly grown over the past years. From the early 1990s to 2000s DDoS attacks have shifted from megabytes to gigabytes (van Rijswijk-Deij *et al.*, 2014). Some researchers consider DoS attacks as one of the most dangerous currently in existence (Alomari, Manickam, Gupta, Karuppayah & Alfaris, 2012).

DoS attacks are also known for stealth attacks, which are arduous to detect. Studies have proven that simple threshold-based statistical anomaly detection methods are not able to detect simple stealth DoS attacks (Aqil, Atya, Jaeger, Krishnamurthy, Levitt, McDaniel, Rowe & Swami, 2015).

The most common stealth attacks identified by Darwish, Ouda & Capretz (2013) are discussed below.

- **TCP SYN flood attacks**
  The attack is based on the TCP protocol and has been very popular within hacking communities. During a normal TCP exchange, the connection starts with a three-way handshake as shown in figure 2.7(a). The typical handshake begins with the user sending a legitimate request to the server in the form of a SYN message. The server responds by sending back a SYN request (SYN-ACK) to the legitimate user. To initiate the third handshake the user sends an Acknowledge (ACK) request to the server to establish the connection.

  The attacker would take advantage of the three-way handshake by flooding the system with requests. The system can practically only maintain X amount of states depending on the resources. The server will pool the requests, and wait to complete the process for all the half-open states (only completed the first handshake). Afterwards, the system will be unable to process further legitimate requests as shown in figure 2.7(b).

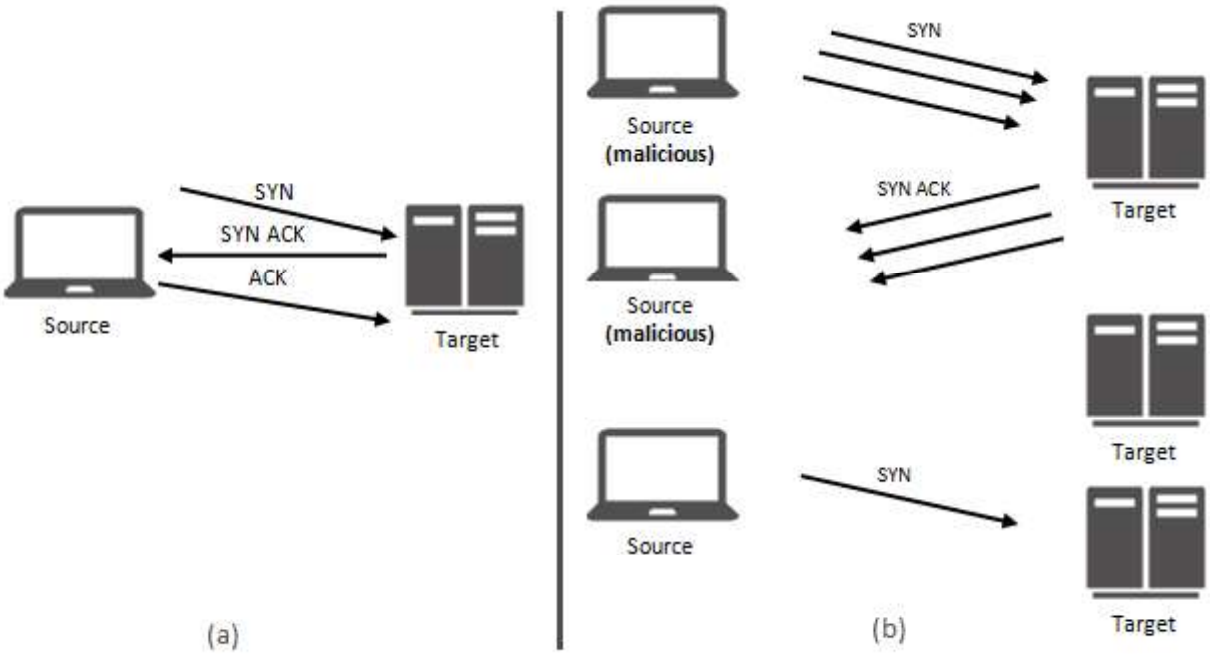

Figure 2.7: SYN flood attack adapted from Luckner (2015)

- **Slowloris** is another TCP-based attack that exploits HTTP (Damon, Dale, Laron, Mache, Land & Weiss, 2012). The attacker will establish multiple connections to the HTTP server and keep the connection transmitting by sending incomplete HTTP headers. Normal network exchanges are illustrated in figure 2.8. The attack exploits the machine's allocatable sockets by keeping multiple connections alive until the available sockets are depleted, a traditional DoS attack *modus operandi*.

The attack is much more resource intensive due to maintaining multiple connections. To decrease resource requirements, the attacker can send HTTP headers in short bursts periodically. The attack can be considered very stealthy due to the sheer difficulty of discriminating between legitimate and Slowloris traffic. The Slowloris attack is illustrated in figure 2.9.

Figure 2.8: Normal network exchange

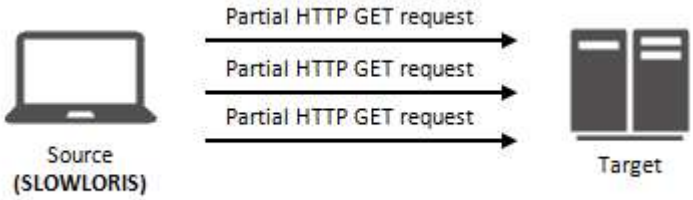

Figure 2.9: Slowloris attack

Combined with probe attacks to identify vulnerable amplifiers, a simple DoS attack becomes hazardous. Amplifiers are any protocols (e.g., NTP, DNS) running on the host that reply with packets, in aggregate size querying packet size. Czyz, Kallitsis, Gharaibeh, Papadopoulos, Bailey & Karir (2014) voice their concerns regarding the rise of DoS attacks using amplification, which causes a steady increase in such attacks. We further look at probe attacks, considered the spawn of advanced attacks, in the next section.

2.3.4 Probes

Information gathering is a crucial part of any hack or penetration test and involves the attacker creating a clear understanding of the system and perimeter being targeted (Wilhelm, 2009). The intruder would scan the system for vulnerabilities scoping the network, hardware and software to be exploited.

Probe attacks can be considered as the baseline for any attack. Therefore, detection is of great importance. Imagine a boxer during the first round of a fight, jabbing away at his opponent, searching for weaknesses. When the opportunity arises, the boxer will start targeting his punches to control the fight. Within the cyber domain, the attacker would scan the network on different layers to collect information about the system and the underlying security controls. It is crucial to detect or completely limit system probing as this is the baseline that leads to more sophisticated attacks. To summarise the characteristics and features of probe attacks within the research data set, table 2.4 is adopted from Kendall (1999) who created the DARPA data set, widely known for intrusion detection. The DARPA data set was adopted to

become the KDD Cup '99 data set (KDD Cup, 2016). The researcher further updated the table to include more modern probe attacks as included in the NSL-KDD data set created by Tavallaee *et al.* (2009) and used within this research project. Refer to Chapter 6 for more information.

Table 2.4: Probe attacks summarised adapted from Kendall (1999) & Tavallaee et al. (2009)

Name	Service	Vulnerable platform	Mechanism	Time required	Impact
Ipsweep	ICMP	All	Feature abuse	Short	Reveals active machines
Mscan	Several	All	Feature abuse	Short	Reveals known vulnerabilities
Nmap	Several	All	Feature abuse	Short	Reveals active ports on one machine
Saint	Several	All	Feature abuse	Short	Reveals known vulnerabilities
Satan	Several	All	Feature abuse	Short	Reveals known vulnerabilities
Portsweep	Several	All	Feature abuse	Short	Reveals machines with specific active port

Probe attacks are often also referred to as port scans. Lee, Roedel & Silenok (2003) classify port scans as follows:

- **Vertical scans**
  The scan, as the name suggests, operates vertically and targets a single host while looking at several ports. For intrusion detection, only single host detection are required to detect the attack, rendering it fairly detectable.
- **Horizontal scans**
  The scan, as the name suggests, operates horizontally and targets several hosts while only looking at specific ports. This is the inverse of vertical scans. This attack is relished by attackers who are already aware of a vulnerability, and are probing for hosts to exploit. Whenever the vulnerability for the specific port is published, the intrusion detection system would be updated specifically for the attack. This is an excellent example of signature-based detection that requires signature updates before being able to operate to its full extent.
- **Block scans**
  The block scans attack is the combination of both horizontal and vertical port scans to sweep the whole address-port spectrum within a target environment.

A recent study by Patel & Sonker (2016) classifies port scans differently, with a focus on the Probe attack's anonymous nature. The probe attack could be considered non-stealth or stealth probes, the latter trying to bypass firewalls, routers and masquerading as normal network activity.

Since we are focusing on detection in this thesis, a closer look into stealth probe methods is required. Singh & Tomar (2015) highlight that current detection techniques are infeasible to properly detect probe attacks. Singh & Tomar (2015) state that probe techniques have become highly distributed, composite and stealthy, hindering detection. Stealth probes do not produce any TCP sessions, remaining untraceable within application logs. Vasilomanolakis,

Stahn, Cordero & Muhlhauser (2016) show that defending against even basic probe attacks require compromises. We can further classify stealth probes as summarised below.

SYN/ACK scan: This attack is fast and avoids using a three-way handshake. The source sends a SYN with ACK flag to the target, expecting a RST packet if the port is closed. Figure 2.10 below visualises the attack and packet response.

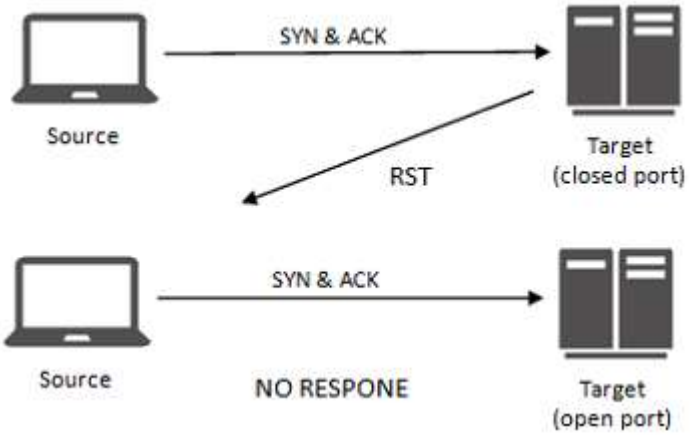

Figure 2.10: SYN/ACK probe attack adopted from Bani-Hani & Al-Ali (2013)

XMAS, NULL AND FIN scan: These attacks are grouped due to the similarities. The attacks would target TCP ports by sending a single frame without any TCP handshake or additional packet exchange (De Vivo, Carrasco, Isern & De Vivo, 1999; Kumar & Sudarsan, 2014).

With XMAS attacks the attacker would send a packet with the FIN, URG and PSH flags set. As with SYN/ACK attacks, if the port is closed, the target will respond with a RST packet as shown in figure 2.11 (Lyon, 2009). The FIN attack, shown in figure 2.12, sends a packet with just the FIN flag set. The same response is expected as with XMAS attacks. NULL attacks send packets without any flags set, expecting the same RST packet response as shown in figure 2.13.

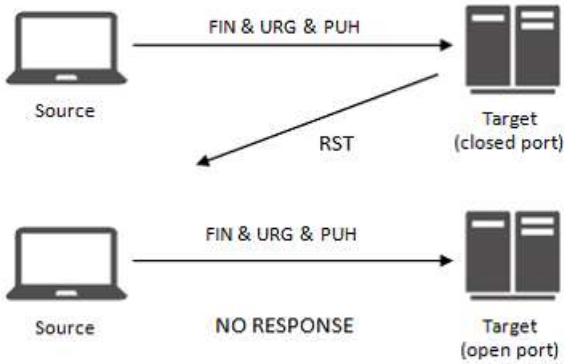

Figure 2.11: XMAS attack

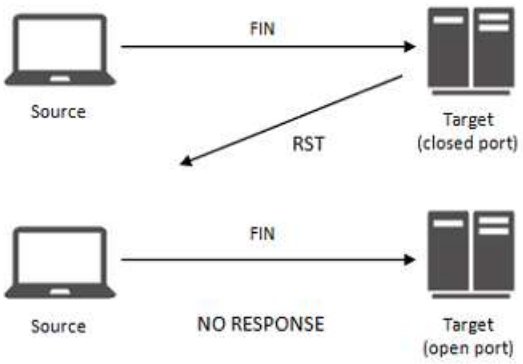

Figure 2.12: FIN attack

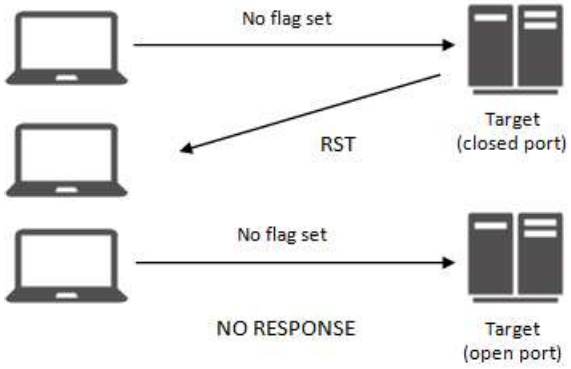

Figure 2.13: NULL attack

Probe attacks can evade packet filters by using TCP fragmentation. This technique works for any of the above methods by first decomposing the packets into fragments (Lyon, 2009). Within intrusion detection systems, a signature that triggers whenever a series of User Datagram Protocol (UDP) packets have been sent to different destination ports from a specific host, would be one of many examples possible for detecting probe attacks. These attacks have been around for several decades, some evolving into more sophisticated models and others exploiting the security measures in place.

When we consider the different layers, we can analyse the attacks on a packet level, identifying the characteristics (*features*) that go hand-in-hand with the attack. We analyse the features as we split the NSL-KDD data set per attack type in Chapter 6. As the research continued and technology advanced, intrusion detection systems that use data collection, data pre-processing, intrusion recognition, reporting and response activities to detect potential and actual computer intruders were created (Inayat, Gani, Anuar, Khan & Anwar, 2016).

In the modern environment, several types of IDS methodologies exist, each with its strengths and weaknesses. In section 2.4 we introduce the methods used to detect the attacks discussed.

2.4 Intrusion detection methodologies

IDS makes use of several methodologies to detect the variances and anomalies. The most used detection methodologies are signature based, anomaly based detection with newer

methodologies as artificial neural network and hybrid methods (Scarfone & Mell, 2007; Modi, Patel, Borisanya, Patel & Rajarajan, 2012).

2.4.1 Signature detection

Signature-based detection uses threat signatures for comparisons and observes events to identify hazardous incidents (Scarfone & Mell, 2007; Inayat *et al.*, 2016). The signatures are compiled from previous attacks and represent the characteristics to identify such an attack. The core functionality relies on pre-processing the captured network packets to find a signature and then compare them with commercial or pre-built definitions (*signatures*) of known malicious content (Scarfone & Mell, 2007; Inayat *et al.*, 2016). Not only does it analyse the packets at a granular level (*i.e. particles*), it also can identify harmful behaviour. Signature-based detection methods could easily be spoofed by using multiple attacks or completely unknown attack vectors not identified within the threat signatures (Scarfone & Mell, 2007).

The system relies heavily on the signature and definition files to be updated with new attacks to be effective. Signature-based detection is one of the most used in commercial products (Han & Cho, 2005). Most commercial IDSs utilise the network and transport layer to build signature definitions (Yegneswaran, Giffin, Barford & Jha, 2005). The operations within a signature-based detection IDS is illustrated in figure 2.14.

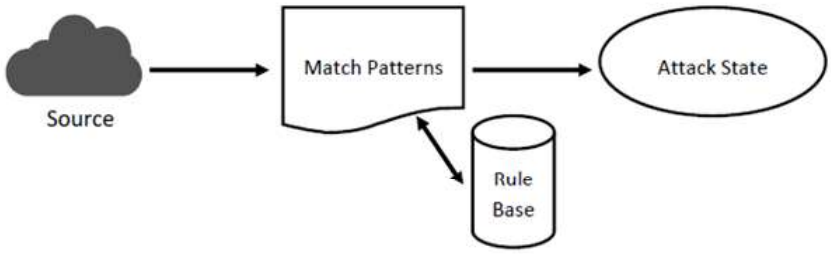

Figure 2.14: Signature-based detection

During reconnaissance (*gathering information of the environment under attack*) the attacker would identify systems that have not been updated with new signatures. This requires the system to have a pre-configured knowledge base and would then require it to be updated continuously. One can imagine that using signature-based detection will be much easier to deploy since the system does not have to learn the environment as in anomaly detection. Singh & Nene (2013) highlight the need for significant resources to keep up with the infinite number of modifications to known threats. They further indicate that signature methodology is simpler to modify and improve, since its core is built around the signatures and rules deployed (Singh & Nene, 2013). We can summarise these characteristics as seen in table 2.5.

Detailed focus is placed on the limitations of this technique in section 2.5 as we look at recent research on intrusion detection from a critical point of view. Features are extracted from the incoming packets and a signature is build from the packet features. When the packet features signature matches the signature in the intrusion database the alarm is raised. The packet features signature is then passed to a function that evaluates whether the signature is new. If

new, the signature will be added to the original signature database. In the event of no match between the packet features signature and the database signature, no alarm will be raised and the system proceeds as normal.

Table 2.5: Advantages and disadvantages of signature-based detection

Advantages	Disadvantages
Accurate against known attacks	Cannot detect new variant of attacks
Low computational cost	Intensive crafting signatures
Easy to deploy	High false alarm for unknown attacks

The processes used with signature detection can be outlined using high level pseudocode as displayed below in the Signature detection pseudocode listing.

```
1  Algorithm signature based detection is
2          input: Incoming packets INP,
3          criteria features CF from database,
4          database signature S
5          output: New signature NS,
6          real alert R
7  //Function receives incoming packets and compares with criteria features
       in database
8  function evaluatePackets(INP)
9    for each INP do
10         Extract the features from INP based on CF as packet features PF
11         Build signature X from PF
12         if (X == S) then
13                 Function raiseAlarm(X) as R
14                 Function addSignature(X) as NS
15         else if (X =! S) then
16                 #No alarm proceed normal
17         end if
18     end for
19 end function
```

Listing 2.1: Signature detection pseudocode

Some of the issues experienced with signature-based detection are overcome using anomaly detection.

2.4.2 Anomaly detection

Anomaly-based detection functions more intelligently, and constantly evaluates definitions of 'normal' activities created by the system against observed events to identify significant deviations. Unfortunately, this method can also generate several false alarms if the intruders manage to create more complex profiles of which certain malicious activities have already been included in the systems' pre-evaluated 'normal' definitions (Scarfone & Mell, 2007). Traditional anomaly detection is widely based on statistical tests, while recent research

focuses on more sophisticated methods with machine learning. These enhancements have been praised by Modi *et al.* (2013).

The general architecture for anomaly detection is shown in figure 2.15. The system dynamically updates the profile and as the user's pattern changes, so does the profile. In comparison with signature-based detection, the 'database' receives updates in a more static way, while the anomaly-based profile generation is an ongoing process.

Some aspects of the users' activity can be modelled precisely. Assume user X logs into the computer at 9 am, performs database transactions, reads e-mails and then takes a break between 1 pm and 2 pm. On average, during the morning activity X generates very few file access and other errors. The system will generate a profile for user X based on these behaviours. If on Sunday at 3 am activity on user X's account is logged with numerous file access errors, that activity would instantly be flagged as suspicious (Modi *et al.*, 2013).

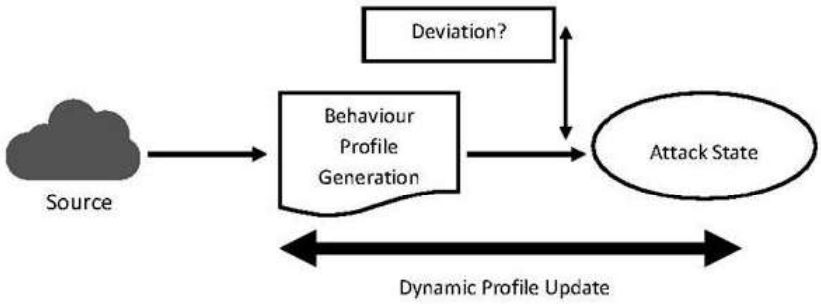

Figure 2.15: Anomaly-based detection

The advantages and disadvantages of anomaly detection are summarised in table 2.6.

Table 2.6: Advantages and disadvantages of anomaly-based detection adopted from Modi et al. (2013)

Advantages	Disadvantages
Lower false alarm rate for novel attacks	Generates more false alarms for known attacks
High computational cost	Detection relies heavily on data collected from system
Does not require prior knowledge able to detect novel attacks	Environment needs to be capable of sourcing data for profiles
Easier to maintain once implemented	Learning process takes time to become accurate and effective
Can detect "low and slow" attacks	Difficult to predict detection sensitivity
Very tailored, normal activity customised for each entity	Difficult for attackers to spoof user activity without being detected

Further analysis of anomaly detection is done in section 2.5 as we look at recent research on anomaly-based intrusion detection from a critical point of view. The processes used within anomaly detection can be outlined using high level pseudocode as illustrated on the next page. The intrusion detection system receives the current network data as test data. The system

then loops through each process within the test data; if the process is a deviation from the normal user behaviour the alarm will be raised. If the process is considered normal, the function calculates the probability of the normal process occurring within normal behaviour. Finally, a K-nearest neighbour algorithm is used to identify any deviation from the user's average normal behaviour.

```
1  Algorithm anomaly based detection is
2          input:  Test data D,
3          normal behaviour data set TD,
4          output: real alert R
5  //Function receives test data, then analyses each process against
6  normal behaviour as well as thresholds
7  function evaluateProcess(D)
8    for each process P in D do
9          if (P =! TD) then //unknown process or system call
10                 Function raiseAlarm(P) as R
11         else then
12                 for each normal process NP in TD do
13                         calculate simProbability(P,NP)
14                         if (simProbability == 1) then
15                         //P is normal
16                         end if
17                 end for
18                 find k biggest scores of simProbability(P,NP)
19                 calculate simProbability_avg for k-nearest neighbors
20                 if (simProbability_avg > threshold) then
21                         //P is normal
22                 else then
23                         Function raiseAlarm(P) as R
24                 end if
25         end if
26   end for
27 end function
```

Listing 2.2: Anomaly detection pseudocode using k-nearest algorithm

Denning (1987) classifies anomaly-based detection into several categories. The classification is supported by several other researchers as well (Qayyum, Islam & Jamil, 2005; Chandola, Banerjee & Kumar, 2009; Gyanchandani, Rana & Yadav, 2012; Devare, Shelake, Vahadne & Tamboli, 2016).

- **Threshold Model:** This model is based on the assumption that an anomaly can be discovered when comparing the observations with a predefined limit, and it builds on the cardinality of observation over a period of time. For example, user X raises several password failures for a short time, and the system will use an event counter to estimate when to raise the alarm.
- **Markov Chain Model:** This model is based on the assumption that an anomaly can be discovered comparing current events with events that preceded it. In conjunction with an event counter metric, the system will determine how common a specific event is by

comparing it with preceding events. Each observation is assigned a specific state and classified within a state transition matrix, which determines the probability of the event occurring.

Consider user X successfully logging onto the computer – during the start-up, several services will run and be logged. The logs contain descriptive information about the services e.g. process ID, session ID, objects accessed, user ID, etc. Every event performed on the computer would include a sequence of actions. The temporary behaviour on the user's machine can be defined as discrete-time stochastic processes. These discrete points in time are not bound by a fixed time interval, but by times when events take place. For the computer user, the actions to take place are related to the last event as well as preceding events. Markov chain models represent a temporal profile of normal behaviour for the events performed by the user on the computer.

The Markov chain, therefore, learns from historical data based on the system's normal behaviour. Whenever a new event is actioned, the behaviour of the computer is analysed to draw a probability that the Markov chain model of the normal profile supports the new behaviour (Ye, 2000). A low probability score indicates anomalous behaviour that might result from malicious activities.

- **Statistical Model:** Within statistics, a moment is a mean or standard deviation (Qayyum *et al.*, 2005). This model is based on the determination of normalcy for an observation and assigns positive confidence within a specific range. Shrewdly, the model focuses on user activity over a period, instead of previous user activities. Whenever an event falls outside the set interval or above, the moment can be considered anomalous in nature. The model also considers system changes by adapting the statistical rule base on which decisions are based (Qayyum *et al.*, 2005). The model trumps operation methods due to not requiring any prior knowledge to determine normal activity and set limits. It is focused on the "moment" by learning normal activities from observing, and the confidence levels adapt to the increased knowledge.

  The added benefit of building confidence levels from observed data is the segregation between users. User Y's behaviour might output different confidence levels than that of user X. The flexibility adds severe complexity to the model and therefore might hinder implementation.

- **Multivariate Model:** The multivariate model can be applied to monitor and detect deviations for a process within an information system. Very similar to the mean and standard deviation model, the core functionality is based on correlations among features. Since two or more features are related, multivariate analysis can be considered more suitable (Gyanchandani *et al.*, 2012).

  Debar, Dacier & Wespi (2000) reveal that the model permits the identification of potential deviations, where the complexity of the situation relies on the correlation between multiple parameters. The technique has been very popular within the manufacturing

process to detect and monitor anomalies (Niaki & Moeinzadeh, 1997). Recent studies used multivariate models to attack DoS attacks with great success (Tan, Jamdagni, He, Nanda & Liu, 2014; Devare *et al.*, 2016).

Within intrusion detection, the model can be useful if the data show that better discriminating power can be obtained from a combination of features rather than individual measures e.g. login frequency, session elapsed time and RAM usage. By focusing on the correlation, the model can detect intrusive events at early stages. First the model requires the event to be processed and in effect delaying the real-time execution.

2.4.3 Neural network detection

Artificial neural network detection methods have the ability to generalise data from incomplete data and then classify the data as either normal or intrusive (Abdlhamed, Kifayat, Shi & Hurst, 2017). Some research have revealed how traditional statistical techniques used for anomaly detection can be replaced by artificial neural networks (Cannady, 1998; Abuadlla, Kvascev, Gajin & Jovanovic, 2014; Al-Jarrah & Arafat, 2015). Neural networks have been applied to the intrusion detection domain since the early 90s with great results (Lunt, 1990; Debar, Becker & Siboni, 1992).

The neural networks have the potential to address many of the problems found within signature and anomaly detection (Cannady, 1998). Artificial neural networks have been researched since the 1960s and were met with several theoretical barriers (Debar *et al.*, 1992; Schmidhuber, 2015).

The neural network learns the characteristics of system users and identifies statistically significant deviation from the established behaviour. This approach requires the neural network to be trained using normal and abnormal data. Depending on the amount of data, such training techniques can take up much time and resources. We can highlight the most concerning drawbacks of neural networks as follows (Dumitru & Maria, 2013; Cannady, 1998):

- Training data – large amount of data required to accurately train the model
- Black box algorithms – the neural network's results and performance are opaque and there is no way to test or understand the results with full confidence

We can identify the neural network approach as a step in the intelligent intrusion detection system domain; unfortunately, it is still limited to the research domain (Ahmad, Abdullah, Alghamdi, Demiralp, Baykara & Mastorakis, 2009).

2.4.4 Hybrid detection

Hybrid detection techniques could be viewed as a combination of two or more of the techniques mentioned above. The system will cumulate the advantages that each of the techniques has to improve efficiency, effectiveness and overall performance (Modi *et al.*, 2013). Engen (2010) mentions that most artificial neural networks combined with signature detection are network-based, while host-based topologies favour anomaly detection with

artificial neural networks. The system proposed by Lunt (1990), called Intrusion Detection Expert System (IDES), combined anomaly and signature detection. IDES can be considered the first expert intrusion detection system to combine both detection techniques.

A hybrid system combining decision trees for signature detection and self-organising maps for profiling normal behaviour combined signature, anomaly and machine learning techniques to classify network data on the KDD Cup '99 data set (Depren, Topallar, Anarim & Ciliz, 2005). Another system proposed by Aydın, Zaim & Ceylan (2008) combined signature and anomaly-based intrusions detection systems. The researchers noted that the overall system improved upon traditional signature detection by utilising anomaly detection to detect unknown attacks.

Although the techniques offer promising benefits when combined, there are still clear limitations. We further analyse the intrusion detection field from a critical point of view within the next section.

2.5 Limitations of intrusion detection systems

We have discussed the intrusion detection field on a very high level, introducing the different topologies and detection techniques available. Wang (2008) identifies several characteristics an intrusion detection system must fulfil. These characteristics are based on the work of Debar *et al.* (1999).

- **Fault tolerance –** the IDS itself must be resistant to attacks
- **Completeness –** identified as the core function of any IDS. The system should be able to detect all intrusion attempts leading to a false negative rate of 0. Such as low false negative rate is unrealistic and exceptionally hard to achieve
- **Performance –** the IDS should be able to function and operate without constraining the network or host
- **Accuracy –** the IDS should not classify legitimate traffic as malicious and *vice versa*. This relates to the false-positives within detection. We further discuss evaluation techniques in Chapter 6
- **Scalability –** the ability to process traffic in real-time is of great importance within IDS. This also relates to the data received from different audit agents within the host topology

The requirements clearly outline why building a good and effective IDS is so difficult. The underlying issues have already been drawn in the problem statement, and current techniques are not coping with the sheer amount of new attacks.

A vague picture of the current intrusion detection limitations has been painted but a more complete picture is needed. Several research papers have scrutinised the current intrusion detection domain (Lee *et al.*, 2002; Sommer & Paxson, 2010; Mohammad *et al.*, 2011; Burdette, 2016). An increase in research papers within the field also outlines the need for improvement (Sommer & Paxson, 2010). We further investigate the issues in the next chapter as we look at the limitations of machine learning within the intrusion detection field.

According to Lindqvist & Porras (1999), network-based IDSs are not able to catch all forms of intrusion; some intrusion does not happen within the network layer, therefore a hybrid system should be used to combine host and network-based attacks. Network-based intrusion detection cannot analyse encrypted data. Host-based systems also fail to observe network activity that might form part of an attack on the specific host (Lindqvist & Porras, 1999).

With a successful U2R attack, the attacker can turn off the IDS agent using the root privileges. For NIDS to be effective the system requires great visibility (Schupp, 2000). This however depends on the network and security design; bad implementation can lead to high false alarms or slow down network performance. For example, connecting the NIDS to a network switch will only allow the system to process traffic directed to it. The switch network scenario is not optimal and limits the system. Placing the IDS is extremely important; not limiting the system or crippling its visibility of the network should be considered (Schupp, 2000; Stingley, 2015).

The allocation and consumption of resources are imperative for an IDS. The system requires scalable operations; without the resources to capture, store and analyse large amounts of data in real-time, the IDS becomes just another offline reporting tool. The sheer packets per second can reduce the IDS' ability to keep up (Schupp, 2000). TCP connections require a three-way handshake to detect attacks. The IDS would need to maintain states for the current active connections.

Hackers can utilise several techniques to cripple or reduce the IDS' abilities. The attacker can blind the sensors by flooding the NIDS until it drops network packets. Such attacks are overwhelming for HIDS. With HIDS, the attacker can launch several attacks simultaneously to each endpoint. The system and administrator would have a hard time determining the source or target of the real attack. Some tools allow the attacker to launch decoy scans that simulate real attacks (Schupp, 2000). It is critical for the system to be fault tolerant due to several techniques available to evade the detection system.

Current IDSs are failing to detect advanced stealth attacks. Using fragmentation, the attacker will send network packets fragmented, breaking up the payload into smaller chunks to reduce detection ability or reduce the ability to compare with the attack signatures. The signature detection model lacks the flexibility as only slight variations in the attack sequence can affect the signature comparison (Sadeghian, Zamani & Ibrahim, 2013; Holm, 2014).

Several tools can be used for evading detection and several techniques such as method-matching, parameter hiding and session splicing are offered (Wang & Hong, 2016). We further investigate the challenges brought by stealth, zero-day and attack variants in the closing arguments of Chapter 3 when introducing machine learning to the intrusion detection equation. Network intrusions are constantly evolving; such is the traffic within the network environment. The infinite variety of attacks and the creativity of hackers would require an extremely dedicated effort to update the signatures in the hope to accurately identify variations

of attacks. It is evident that security researchers and security professionals operate within a tough, complex field; the malicious efforts are ever so slightly edging past our endeavours. The slightest improvement regarding accuracy, performance or even practicality pushes the cyber defence domain towards levelling the battlefield. The potential within the machine learning field warrants further investigation and research.

CHAPTER 3

MACHINE LEARNING

In Chapter 3 we look at the machine learning domain from an intrusion detection perspective. Section 3.1 introduces the field of machine learning and data mining. We then move on to the machine learning classification task in section 3.2. Ensemble techniques for machine learning is discussed in section 3.3. The benefits and limitations of machine learning within intrusion detection are briefly discussed in section 3.3 and 3.4.

3.1 Introduction

Data mining combined with machine learning is defined by Parihar & Tiwari (2016) as processing data to gain the implied, prior unknown potential of useful information, which can be expressed as patterns. The previous chapter highlighted the issues and need for improved intelligent components within the intrusion detection domain. To understand the potential of intelligent improvements, one has to understand the different types of intelligence in computing.

Computational intelligent systems as defined by Bezdek (1993) can be identified when it deals with e.g. only numerical data; incorporates pattern recognition components; does not use knowledge in the artificial intelligence sense and progressively, when it exhibits computational adaptability; computational fault tolerance; human-like turnaround speed and failure rates compared to approximate human performance.

Artificial intelligence provides important and comparatively low-cost techniques for designing IDSs while taking care of the energy consumption (Alrajeh, Lloret & Loo, 2013). Artificial intelligent systems are more concerned with improving known algorithms by utilising problem-solving techniques similar to those used by human beings (Frank, 1994). Within this thesis, we lightly cover some intelligent systems such as decision trees and swarm intelligence in the next chapter. Table 3.1 represents the main differences between computational intelligent systems and well known artificial intelligent systems.

Table 3.1: Intelligent system comparison by Craenen, Eiben (2002) & Crosbie et al. (1995)

Computational Intelligence	Artificial Intelligence
Handles numerical representation of information	Handles symbolic knowledge representation
Low-level cognitive functions	High-level cognitive functions
Bottom-up analysis: structure is expected to emerge from an unordered beginning	Top-down analysis: analyses structure of a given problem and attempts to construct an intelligent system based upon the structure

Machine learning is a branch of artificial intelligence; in essence it is the science of giving machines the ability to learn and adapt. Doing so requires intensive use of mathematics and statistics to model the problem and then rely on the computational power of computers to crunch the numbers. However, this would all be in vain if the machine cannot learn. The most popular learning techniques are either supervised or unsupervised (Balon-Perin, 2012).

Consider a generic framework for machine learning where there are data input, a machine learning algorithm and output produced. Supervised learning requires the data input to be labelled, i.e. each entry within the input data set has a label indicating which class the entry belongs to. For intrusion detection, this would require each network record to be already labelled as an attack or normal network traffic. The classification task is further discussed in subsection 3.2.

Consider a child who has to learn to recognise an object with the help of her parents. The parent lines up several objects and tells the child the word corresponding to the object. The child now considers several features of the object such as shape, colour, texture, etc. to remember. With machine learning, these would all be features within the data set. The child will then associate a label (the word told by the parent) to the corresponding object in her memory. When the child observes the object again, she will be able to utter the word corresponding to the label.

Supervised machine learning works similarly. The algorithm builds a model based on the input data set's features and the necessary label assigned to each entry. This also becomes the main disadvantage of supervised learning where the algorithm is unable to learn without a labelled data set. As discussed in Chapter 6, labelled data sets are a big issue within the IDS domain (Sommer & Paxson, 2010).

Unsupervised learning, however, does not require any label within the data set to learn, due to drawing conclusions from the similarities in data as "moments" (Parihar & Tiwari, 2016). This is extremely beneficial, but as one can imagine, it will not provide the same accurate results as supervised learning because unsupervised learning has no grounded truth.

Witten, Frank, Hall & Pal (2016) summarise the following popular machine learning techniques as follows:

- **Regression** – very popular for estimating numerical values such as housing prices, product prices, stock prices, etc. Several techniques of regression exist such as Gaussian process, linear and kernel. The technique is supervised as it requires prior knowledge and data sets labelled to perform predictions
- **Clustering –** the task is all about grouping data and a label associated with each of the groupings together. This is a popular unsupervised approach where the model identifies common similarities in the data, and then combines them. Clustering involves algorithms such as K-means, mean-shifts, etc.
- **Classification –** this technique is very simple and involves classifying unknown data

based on learning from a labelled data set. Although most techniques involve using supervised learning such as decision trees and naive Bayes, this is not always the case. Neural networks can learn unsupervised and still classify data accurately, especially with modern advances in deep learning

It is important to understand that within machine learning, there is no silver bullet. The techniques used to solve a problem need to be tailored to fit, and regression might address the problem predicting housing prices with high accuracy, but will however not add much value to data that are nonlinear. In this thesis we focus on the task of classification although other machine learning techniques might be applicable to an IDS. The underlying ATM algorithm classifies data using supervised learning.

3.2 The classification task

Classifiers receive labelled data from training data sets, align it to groups predefined by their specific qualities and then output a classifier that can predict the class for new items. Classification of intrusion data is a mature research area, and several different classifiers have had success within detecting intrusions (Nguyen & Choi, 2008; Tsai, Hsu, Lin & Lin, 2009).

The purpose of the classification task is to classify data by learning several features from an input data set (Schapire, 2015). The machine learning classification task can be assigned too many modern data problems such as:

- Text categorisation – spam filtering or simple sentiment analysis
- Fraud detection - Classify high risk transactions or banking events
- Machine vision – face detection or object classification e.g. identify weapons within a picture or detect known criminals using facial recognition and surveillance techniques
- Market segmentation – classify clients who will respond to the promotion
- Bioinformatics – classify proteins according to their function

Classifying data using machine learning is more data driven than human-crafted rules, leading to more accurate results in the classifier (Kotsiantis, 2007). Machine learning also removes the human element, and although the benefits have become a heavily debated topic, it still results in a reduction of human errors and consistency when computing results (Gutzwiller & Reeder, 2017).

Machine learning tasks must be fitted and tailored to function at acceptable levels. Before classifying data with machine learning, some requirements must be met (Witten *et al.*, 2016):

- Easy accessible data source – to build a training data set, a data source needs to be accessible and easy to extract data from. The quality of data extracted also needs to be of high integrity as the model is built from it. Frequently, it is hard to obtain enough and high-quality data with current legal regulations, privacy considerations and technical skills required. This increases challenges for sourcing machine learning data substantially
- Labelled training data – training data needs to be very similar to the test data for accurate

models. Therefore the training data needs to be well crafted and accurately labelled. Labelling data can become a tedious task as each entry needs to be labelled, the data within the training data set also needs to be sufficient to build a machine learning model. If labelling the data is not possible, other machine learning techniques can be applied

- Model evaluation – machine learning has not yet evolved into a plug and play data tool (Brink, Richards & Fetherolf, 2016). Several techniques can be used to evaluate machine learning models, and the model needs to be correctly evaluated. The dimensions within the training, evaluation and test data sets are crucial to understand. The value of evaluating the model depends on the type of data used to create it. This is seen within this thesis as well, in Chapter 6, where the performance metrics are discussed. For example, highly unbalanced data sets might show high accuracy, however, the metric cannot be relied on when using unbalanced data sets and the f-measure would be a better representation on how the model performed
- Model testing – test data need to represent real situations. If this is not the case, a clear discrepancy between test results and actual implementation performance will be noted. Data simulating real networks are among the most difficult to label correctly

It is clear that classifying data is no easy task. Often when machine learning is tasked with real world problems, massive amounts of data need to be worked through. We have established the importance of data quality and the "garbage in, garbage out" methodology can haunt accurate results with machine learning (Yang, 2010).

Data pre-processing has become a crucial part of any machine learning task. Kotsiantis, Kanellopoulos & Pintelas (2006) state that the representation and quality of instance data are the most important factor within machine learning tasks. Data pre-processing is a process where data are cleaned up, normalised or relevant features are extracted (Kotsiantis *et al.*, 2006). The process occurs as the final training step and allows for improved performance and accuracy.

Feature selection has become a useful technique within the pre-processing process to improve machine learning success. Selecting useful features will reduce data dimensionality and lead to improved performance as well (Dinakaran & Thangaiah, 2013). Kotsiantis *et al.* (2006) explain the feature selection process as identifying and removing as much irrelevant and redundant features as possible within a training data set. With fewer features to process, machine learning algorithms operate quicker and more efficient. Kotsiantis *et al.* (2006) characterise features within a data set as follows:

- Irrelevant – features that do not have any influence on the result
- Relevant – features that affect the result and their role cannot be assumed by any of the other features
- Redundant – features which role can be assumed by another feature

A myriad of feature selection techniques exist. In this thesis, the information gain and ranker techniques were used. The information gain technique measures the information in bits with

regard to the class prediction (Hall & Holmes, 2003). It measures the uncertainty associated with a random feature. If the feature produces zero information gain, it does not add value to the class prediction and can be removed.

Utilising the ranker search method, each feature is ranked based on the feature evaluator's result e.g. information gain, entropy. The ranker method provides a rating for the feature, ordered by its score in relation to the information gain (Dinakaran & Thangaiah, 2013). We further discuss how features were selected using WEKA in Chapter 6, followed by a discussion of the results in Chapter 7. Although machine learning results can be improved using feature selection, ensembling can also be applied.

3.3 Ensembling

Dietterich (2000*a*) notes that ensembling techniques are the combination of one or more machine learning algorithms that constructs a set of classifiers and then classifies data based on the overall popular predictions. To simplify, ensembling produces a new classifier combining individual decisions from several classification models. An example of a classification prediction model is provided in Appendix C, table C.2.

The individual models can be combined based on a weighted voting, averages or unweighted voting (Dietterich, 2000*a*; Gatta, Vallati, De Bari, Pasinetti, Cappelli, Pirola, Salvetti, Buglione, Muiesan & Magrini, 2014; Lacy, Lones & Smith, 2015). The key ingredient for creating a useful ensemble classifier is combining several diverse and accurate models (Hansen & Salomon, 1990). Ensemble classifiers can be created by using the following techniques:

1. **Bagging:** Manipulating the training data set by training several prediction models based on only a sample of the training data drawn randomly (Breiman, 1996; Alfaro, Gamez & Garcia, 2013; Lipitakis & Kotsiantis, 2015).
2. **AdaBoost:** Manipulates the training set to create several hypotheses. AdaBoost uses a weighted function of each training set to reduce the error rate (Freund & Schapire, 1995; Lipitakis & Kotsiantis, 2015). We can deduce that more weight will be placed on misclassified training sets and less on a set correctly classified, thereby progressively improving the result.
3. **Randomness:** The method for implementing randomness into classifiers is highly dependent on the type of classifier used (Dietterich, 2000*a*; Lipitakis & Kotsiantis, 2015). For decision trees, Dietterich (2000*b*) implemented randomness by randomly selecting the top value sets for splitting candidate decision trees.

Ensemble techniques are a matured research area and in most cases lead to higher accuracy in classifiers (Dietterich, 2000*a*; Banfield, Hall, Bowyer & Kegelmeyer, 2007). We ensemble the ATM classifier in a similar method to bagging as discussed in Chapter 6. In the next section, we investigate the benefits of applying machine learning within the intrusion detection field.

3.4 Benefits of machine learning within intrusion detection

In the previous chapter, we have established the requirements for intrusion detection systems and now investigate how machine learning can bridge the gap between the current state of intrusion detection systems and the requirements for a optimal functioning IDS.

As with any intensive data task, intrusion detection systems are required to process large sets of data. Machine learning within intrusion detection can assist in extracting the required information from unknown data sets and identify irregularities (Sommer & Paxson, 2010; Holm, 2014). Regrettably, zero-day exploits or novel attacks, attacks not previously known, are still among the most dangerous (Zetter, 2014). Signature detection systems battle tremendously with novel attacks as discussed in Chapter 2 on page 25.

The problem worsens as attacks spawn several variants, which might not always be added to the signature database timeously. Intelligent detection techniques built from machine learning models would be able to learn and adapt much quicker than waiting for updates from service providers and then the deployment thereof within a company (Buczak & Guven, 2016). Some machine learning techniques generalise much better with data and have therefore more accurate results.

The real power of machine learning can be harnessed from anomaly detection techniques, as this does not depend on updating any signature (Omar, Ngadi & Jebur, 2013). The core of anomaly detection is built upon learning normal user behaviour, as learning is what machine learning algorithms do. This highlights another potential benefit for peeking into machine learning techniques.

The accuracy and fault tolerance of machine learning models mainly depend on their error rate; in several machine learning techniques the error calculation or sensitivity can be tuned (Fawcett, 2006). This allows for a deeper sense of balance between accuracy and fault tolerance. Machine learning techniques can process data fast, adapt and learn (Buczak & Guven, 2016). This nudges another benefit as machine learning techniques capable of functioning with real-time network data would be perfect for intrusion detection systems. The vast amount of data flowing through a network requires quick processing. Machine learning also becomes useful in offline detection; one can argue that the vast amount of data harboured can be dissected by machine learning techniques without impacting network performance or production capabilities when implemented in a segregated environment. Although intrusion detection systems require real-time detection, the model can be trained offline and still be used to detect intrusions in real-time. With all the potential benefits, one might ask, why are we not there yet? Sommer & Paxson (2010) highlight the shortcomings of intrusion detection systems and the effect of machine learning research within the domain. Machine learning techniques have weaknesses that need to be overcome to build efficient or useful intrusion detection systems (Wu & Banzhaf, 2010). Combining the two domains are complex and demands data of high integrity in order to do so. The next section highlights some of the weaknesses within machine learning and intrusion detection.

3.5 Challenges of machine learning within intrusion detection

The limits of intrusion detection systems were briefly discussed in the closing arguments of Chapter 2. Research performed by Sommer & Paxson (2010) indicates the differences between the IDS domain and areas of machine learning, claiming that the problem originates within the premise and that anomaly detection is suitable for finding novel attacks and does not hold with the generality commonly implied.

Sommer & Paxson (2010) further state that the advances and potential of machine learning tools are within finding an activity that is similar to something previously seen, without identifying the activity's characteristics up front. Their statements refer to signature and anomaly-based intrusion detection systems, promoting the concepts on which signature-based intrusion detection systems are founded.

Research has shown there is a clear "success discrepancy" within the intelligent intrusion detection research domain, due to the domain exhibiting characteristics that would make effective deployment of machine learning approaches much more complicated and harder than in other contexts where the specific machine learning methods were successfully implemented (Sommer & Paxson, 2010). Research performed by Wu & Banzhaf (2010) agree that current intrusion detection systems had shown limitations within the requirements of achieving high detection accuracy and "on the fly" processing speeds when confronted with the adaptability requirements of intelligent intrusion detection. A set of guidelines aimed at improving the current research is provided by the two authors.

First of all, intrusion detection is very different from the other domains in which machine learning has been applied successfully. Intrusion detection demands much more real-time detection and processing capabilities to handle massive amounts of data and highly accurate results. In most instances, the cost of investigating false positives outweighs the cost of breach (Sommer & Paxson, 2010). Spam detectors are a very good example of this. Machine learning has been extensively applied within the domain, however, the level of precision required is much lower than with intrusion detection systems (Guzella & Caminhas, 2009). We discuss the requirements for intrusion detection research in the closing section of Chapter 6.

Gollmann (2010) states that most commercial intrusion detection systems use signature detection. Several researchers find the false negative rate within signature detection alarming, specifically when faced with unknown attacks (Kumar, 2007; Gollmann, 2010; Hoque *et al.*, 2012; Aghdam & Kabiri, 2016). Signature detection also favours low false positive rates. Engen (2010) argues differently, stating that this general perception is not accurate for intrusion detection systems (Liao *et al.*, 2013). The application of machine learning techniques within intrusion detection allows signature detection to be more flexible and detects larger variations of attacks (Engen, 2010).

Sommer & Paxson (2010) disagree, noting that there is a certain lure to the potential benefits; however, the problem originates on the premise that the task of finding attacks with machine learning is similar to other applications of machine learning. Despite the popular uptake of

intrusion detection research combining machine learning, the techniques are rarely employed in ”real world” environments (Sommer & Paxson, 2010).

It is evident that intelligent components for anomaly detection would still produce too many false positives, rendering the cost of implementing a pure anomaly detection system too high. The traditional methods of signature detection are still the most widely adopted detection system due to the high accuracy and simplicity (Hubballi & Suryanarayanan, 2014; Holm, 2014; Raiyn, 2014).

Signature detection is extremely effective only if the exact characteristics of the attacks are known beforehand. The method suffers from the inability to identify attacks that occur over an extended period or unknown attacks (Wu & Banzhaf, 2010; Holm, 2014) .

A division of an attack over time or among seemingly unrelated attackers are very difficult to detect using signature detection methods. With signature detection, the assumption is made that the attacks can be precisely encoded in such a manner that identifies variations of the activities, which exploits the vulnerability (Kumar, 2007; Holm, 2014). The assumption is flawed, as attacks grow more sophisticated, more rigid and stealthy, rendering an IDS useless. It is easy to identify a pattern within the intrusion detection domain, a new premise based on recent research advances (Kumar, 2007; Sommer & Paxson, 2010; Buczak & Guven, 2016).

Signature detection limitations can be overcome by implementing anomaly detection. Delusional at best, it can be noted that this creates more problems than it solves (Sommer & Paxson, 2010; Wu & Banzhaf, 2010; Buczak & Guven, 2016). The premise is flawed, as there is a trade-off between detecting novel attacks and raising several false alarms as you adjust the sensitivity of an algorithm (Sommer & Paxson, 2010; Fawcett, 2006). The unfortunate fact is that most breaches monetarily amount to less than losses incurred from detecting false positives (Foster, 2015; Ponemon Institute, 2015; Williams, 2016). It is, therefore, a “catch 22” situation for the intrusion detection field. Thus, although intrusion detection systems need to be able to adhere to the set of requirements, we first need to be able to apply anomaly detection accurately.

This chapter tugged the intrusion detection domain closer with machine learning. The ATM is built upon decision trees and ant colony optimisation, both popular machine learning techniques as discussed in the chapters to follow.

CHAPTER 4

INTELLIGENT CLASSIFIERS

In Chapter 4 we discuss the two pillar machine learning techniques that support the ATM classifier. In the first section (4.1) we discuss a popular machine learning technique called decision trees. Section 4.2 introduces a more complex field of machine learning called swarm intelligence, and we specifically discuss ant colony optimisation and its implementation within other domains.

4.1 Decision trees

4.1.1 Introduction

Decision trees can be used for classification of categorical variables or the prediction of continuous variables. Decision trees are tree-like graphs consisting of internal nodes that represent a test of an attribute, branches that denote the outcome of such tests, and leaf nodes that outline the label. The path followed from the root node to the leaf indicates the rules for classification. The main advantage of decision trees over other classification algorithms is that they provide a rich set of rules that are easy to understand and can be integrated with real-time technologies. Singh & Nene (2013) note that although decision trees are very accurate, they are computationally intensive on large data sets. Figure 4.1 illustrates a simple decision tree that classifies intrusion data.

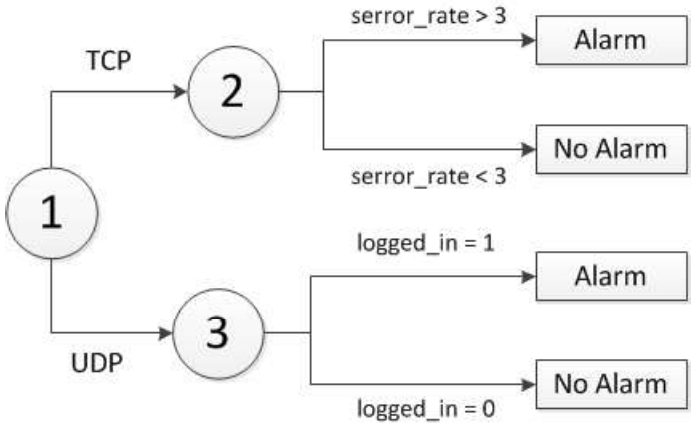

Figure 4.1: Decision tree example

The protocols TCP and UDP represent branches of the root node 1. The two classifications, $serror\_rate$ and $logged\_in$, are internal nodes. TCP protocol with a $serror\_rate$ value higher than three will trigger an alarm. The discrete function of each input attribute determines the split for each internal node.

As with the figure above and in most cases with each test considering a single attribute, each

instance space is then partitioned according to the attribute's value. Although simplistic, decision trees can quickly become complicated, numeric attributes can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of the axes (Rokach & Maimon, 2015).

Breiman, Friedman, Stone & Olshen (1984) note that the complexity of the tree greatly influences the accuracy. We further investigate this statement in Chapter 6 as we create a cost metric for the ATM classifier. We can, therefore, measure the tree complexity using the total number of nodes, the total number of leaves, the tree depth and the number of attributes used (Rokach & Maimon, 2005). Breiman *et al.* (1984) state that tree complexity is controlled by the method used to stop and prune the decision tree. We further discuss techniques used for decision trees in the next section.

4.1.2 Techniques for decision trees

In this section we briefly introduce the steps taken to build a decision tree. Finding optimal decision trees for a training data set is an NP-hard problem (Rokach & Maimon, 2015). Consequently the same applies to finding a minimal equivalent decision tree or building optimal decision trees (Rokach & Maimon, 2015).

We can divide the heuristic methods required to solve the problem in two groups: bottom-up and top-down. The latter group has become more popular and includes inducers such as ID3 created by Quinlan (1986), C4.5 by Quinlan (1993) and the Classification and Regression Trees (CART) developed by Breiman *et al.* (1984). Although old, these techniques are among the most popular used for decision tree induction (Kotsiantis, 2013). Some techniques create decision trees in two phases, growing and pruning (C4.5 and CART).

We can classify the steps to build decision trees as splitting, stopping, pruning and inducing, based on the outlines given by Rokach & Maimon (2015).

Splitting methods: Within each iteration the algorithm building the decision tree would need to consider the partition of the training data using the outcome of a discrete function of input attributes (Rokach & Maimon, 2015). Selecting the most appropriate function is based on the values used to determine the split.

- The criteria can be based on the origin of the measure – information theory, dependence and distance
- The measure structure – impurity-based criteria, normalised impurity based on criteria and binary criteria

Stopping methods: Once the nodes are divided based on the split criteria, the algorithm continues to subdivide the training data until no further splitting satisfies the criteria or until the stopping criteria are satisfied. The condition for stopping is based on the following common rules (Rokach & Maimon, 2015):

1. Instances in the training set belong to a single value of y.

2. The maximum tree depth has been reached.
3. The number of cases in the leaf nodes is less than the minimum number of cases for the parent nodes.
4. Once the node is split, the number of cases in one or more child nodes would be less than the minimum number of cases for the child nodes.
5. The best splitting criteria are not greater than a predetermined threshold.
6. If any of the conditions are met the algorithm would stop. These rules are only common rules and some algorithms might contain more or less rules.

Pruning methods: If the stopping technique used is too strict, the decision trees created will be small and underfitted. Underfitted decision trees are built when the decision tree model is too simple and does not fit to the data well enough. Overfitted decision trees are too complicated and large. Over- or underfitted decision trees both create prediction models that cannot be relied on fully. Breiman *et al.* (1984) first suggested the use of pruning to reduce the complexity and size of decision trees (Rokach & Maimon, 2015).

Popular techniques for pruning are performed top-down or bottom-up. The nodes are pruned if the result would improve a certain heuristic or meet criteria. We can summarise the following pruning techniques noted as the most frequently used when building decision trees by Rokach & Maimon (2015) and Barros, De Carvalho & Freitas (2015):

- Cost – complexity pruning measures the average error reduced per leaf to prune the decision tree (Barros *et al.*, 2015). The number of errors for each node is calculated if collapsed to leaf and then compared to the errors in the leaves. For example, to prune a tree G in a node g means that g becomes a leaf node and all descendants of g are removed. It is important to note that the technique does not consider all pruned subtrees, but only the best ones (Barros *et al.*, 2015)
- Minimum error pruning proposed by Clark & Niblett (1986) is performed from the bottom-up and seeks to minimise the expected error rate for masked cases. The technique is built on determining the importance of a *priori* probability on the estimation of the error, and this is done by using a parameter m – the higher m, the more severe the pruning (Barros *et al.*, 2015). The value of m should be set based on the discretion of the domain expert and therefore results using this technique can vary
- Error-based pruning proposed by Quinlan (1993) is the default pruning technique used in his C4.5 technique. The technique is performed in a bottom-up fashion. Replacing a non-terminal node by a leaf is not easy; the technique considers grafting a subtree onto the parent's place or to not prune at all. The pessimistic estimate of the expected error is calculated by using an upper confidence bound (Barros *et al.*, 2015). For deciding whether to replace a non-terminal node by a leaf (subtree replacement), to graft a subtree onto the place of its parent (subtree raising) or not to prune at all, a pessimistic estimate of the expected error is calculated by using an upper confidence bound
- Reduced error pruning proposed by Quinlan (1987) utilises a pruning set to evaluate

the goodness of a given subtree T. The classification error in the pruning set is used to evaluate each non-terminal node, $T \epsilon \zeta_T$. The conditions for pruning are based on the decrease of error – only when the error decreases will T be pruned. A constraint put in by Quinlan (1987) forces the method to be performed bottom-up; the node t cannot be pruned if it contains a subtree that yields a lower classification error in the pruning set. The method does not work well with smaller data sets (Barros *et al.*, 2015)

Induction methods: We can build decision trees by simply constructing a decision tree based on the training data. The induction method does not allow for generalisation within the decision tree model. Several techniques are used to induce decision trees, and the ATM classifier can be ranked among the latest. Let us first take a step backwards and introduce the more common induction techniques:

- ID3 is one of the oldest decision tree induction techniques created by Quinlan (1986). The information gain is used as a splitting criteria, and the decision tree stops when the information gain is smaller than zero. The ID3 technique does not utilise pruning or handle numeric and missing values
- C4.5 is a welcome improvement on the ID3 technique created by Quinlan (1993). The gain ratio is used as a splitting criterion and halts once the number of instances to be split is under a predetermined threshold. Unlike ID3, the C4.5 technique utilises error-based pruning after the growing phase. The technique handles numeric and missing values as well
- CART created by Breiman *et al.* (1984) constructs binary decision trees; therefore each internal node has only two outgoing edges. The Twoing criteria are used for splitting, and complexity pruning is used to prune the decision tree. CART can handle misclassification costs during the tree induction. The created regression trees use the leaves to predict real numbers instead of class attributes

By following the above-mentioned steps, any decision tree model can be built. The techniques mentioned within the thesis are by no means a comprehensive list as several variations can be combined. A survey study by Kotsiantis (2013) noted that these techniques are among the most popular used in data mining. We look at the implementation of decision trees within intrusion detection later in the chapter. Now that we understand the fundamentals that form a decision tree, let us discuss some of the benefits.

4.1.3 Benefits of decision trees

The following advantages of decision trees have been identified based on the literature in the previous sections:

- Simplistic – decision trees are simple and self-explanatory; we can even convert a decision tree to a set of rules (Rokach & Maimon, 2015). The simplicity allows readers without much knowledge of decision trees to use and understand a decision tree model
- Input variety – decision trees can handle both nominal and numeric input attributes, unlike some neural network algorithms that only use numerical attributes (Rutkowski, Jaworski, Pietruczuk & Duda, 2014). Input variety allows for a broader set of data that can be fed

into the algorithm and vastly improves the popularity

- Rich representation – decision trees can represent any discrete value as a classifier. Decision trees can also handle both continuous and categorical variables fairly well (Rutkowski *et al.*, 2014; Rokach & Maimon, 2015)
- Generalisation – depending on the induction technique used, decision trees can handle missing values and data sets that have errors (Bramer, 2016). This allows once again for a greater audience of data that could potentially be applied. Although decision trees can handle missing values, it severely impacts the accuracy and complexity as discussed in the limitations to follow

4.1.4 Limitations of decision trees

The following limitations of decision trees have been identified based on the literature in the previous section. According to Friedman (1996), most issues with decision trees can be split into two categories: algorithmic problems and representation problems. With the latter, subtrees are duplicated in disjunctive concepts and in many cases lead to the partitioning of data into smaller fragments. According to Friedman (1996), these issues place decision trees at a disadvantage when tasked with classifying data with many relevant features. The problem worsens when decision trees are tasked with handling missing data as the correct branch to take is not known, and the algorithm must use a special technique to handle the missing values (Friedman, 1996). In the case of the C4.5 induction algorithm, the information gain will be heavily penalised (Elomaa, 1994).

Depending on the method used to induce the decision tree, some instances might weigh differently than others (Rokach & Maimon, 2015). This makes it harder to determine the most accurate model and makes certain induction techniques sensitive to data. Decision trees will struggle to classify certain elements with a limited number of training examples. A good example of this is found and discussed in Chapters 6 and 7 as the decision tree heavily leans towards the majority class when classifying. Depending on the complexity and induction methods used, decision trees can be computationally extremely expensive to train. This is investigated in depth by Latkowski (2003) who specifically notes the higher cost when introduced to missing values. Decision trees do not handle non-rectangular regions well. Some algorithms only utilises a single field at a time, which leads to rectangular classification boxes that do not replicate with the distribution space in the data set (Rokach & Maimon, 2015).

4.1.5 Related work within intrusion detection

A long history of implementing decision trees within intrusion detection has been established. We have looked at some of the recent implementations done in the field. Albayati & Issac (2015) and Tavallaee *et al.* (2010) note that decision trees are regarded to be the most reliable and accurate intelligent classifier implemented with intrusion data.

Although decision trees can generalise unseen data, it does not deal well with the missing data. New attacks may be classified as some default class such as ‘normal’, as with the C4.5 decision tree classifier employed in an investigation by Bouzida & Cuppens (2006).

Consequently, this causes false negatives. Therefore, Bouzida & Cuppens (2006) developed a modified C4.5 decision tree classifier that classifies new/unseen data as a new 'unknown' class. By doing this, they avoid a significant amount of misclassifications for new attacks as normal connections, particularly U2R attacks.

Ohta, Kurebayashi & Kobayashi (2008) also propose a modification to the C4.5 decision tree classifier, aimed at reducing the false positive rate. The researchers adapted the way decision trees are built by taking into account the type of errors that may be produced and choosing attributes that are less likely to produce false positives. The modified decision tree classifier is evaluated on subsets of the KDD Cup '99 data set, and is compared with the original C4.5 decision tree. As an alternative approach to reducing the false positives, they also examined random oversampling and undersampling of the training data. The modified C4.5 decision tree classifier outperformed the original decision tree classifier and the sampling approach. Over/undersampling led to similar detection rates, but obtained a much higher false negative rate.

Tavallaee *et al.* (2009) applied decision trees, random forests and several of the most popular classifiers in their NSL-KDD data set, providing a baseline result for the data set. It is important to note that Tavallaee *et al.* (2009)'s research can be used as a comparative, as the same training and test data sets are used. Refer to Chapter 7 for results analysis.

Rai, Devi & Guleria (2016) propose the creation of a decision tree classifier in which they use their split classifier for implementation within intrusion detection. The approach follows a similar methodology as the research described in this thesis utilising the NSL-KDD data set as well.

Machine learning ensemble methods are used to obtain better predictive performance than would have been impossible with any constituent learning method e.g. C4.5, tree learners, decision tree learners and Bayesian methods (Dietterich, 2000*a*). Tesfahun & Lalitha Bhaskari (2013) used cross-validation test techniques, random forests as well as their Synthetic Minority Oversampling Technique (SMOTE) to classify intrusion detection data within the NSL-KDD data set. Their experiments show a reduction in building time and an increased detection rate for minority classes within the data set.

Aggarwal & Sharma (2015) trained ten classification classifiers using WEKA and the KDD Cup '99 data set. Their research highlights a clear dominance for tree-based classifiers, as the random tree classifier shows the best results.

Decision trees have a clear place within the intrusion detection domain, and the implementation thereof can be considered matured (Wu & Banzhaf, 2010; Buczak & Guven, 2016). Research even traces back to the late 1990s as Kruegel & Toth (2003) and Lee, Stolfo & Mok (1999) implemented a decision tree classifier within the popular SNORT intrusion detection system. It is evident that the amount of data has since grown significantly and fallen behind many of the modern ensemble techniques. In the next section, we investigate an

optimisation technique that is ultimately combined with decision trees in Chapter 5.

4.2 Ant colony optimisation

4.2.1 Introduction

Swarm intelligence, a field of artificial intelligence, attempts to replicate the nature of swarms and colonies. Several such research avenues exist such as (Rani & Singh, 2017; Prakasam & Savarimuthu, 2016):

- Ant colony optimisation
- Particle swarm optimisation
- Artificial immune systems
- Bacterial foraging (e.g. biomimicry)

The foraging behaviour and indirect communication patterns of ants have galvanised several optimisation algorithms for NP-hard problems (Prakasam & Savarimuthu, 2016). Ants exhibit fascinating behaviour when foraging for food: they leave a chemical substance, pheromones, to mark a specific route. Pheromones evaporate slowly over time. The strength of the pheromone level will evaporate faster on a longer path because it takes longer to traverse. Thus, shorter paths are chosen more often and build up higher pheromone levels than longer paths. Based on this knowledge, Ant Colony Optimisation (ACO) algorithms are known for solving optimisation problems and clustering data and have been particularly successful when applied in business, engineering and science (Prakasam & Savarimuthu, 2016).

ACO can be considered a metaheuristic that creates and identifies candidate solutions for a given problem. ACO has been used to solve several problems such as (López-Ibáñez *et al.*, 2017):

- Continuous optimisation problems
- Multi-objective problems
- Dynamic problems

The first type of ant-based algorithm created by Dorigo & Gambardella (1997) introduced a new way of classifying data using the nature of ant colonies. This was later extended as the ACO algorithms known today (Dorigo, Birattari & Stutzle, 2006; Rani & Singh, 2017). In order to construct the solution, heuristic information, gathered from the problem instance and artificial pheromone trails, and modified based on the search performance is used (López-Ibáñez *et al.*, 2017). The ACO developed by Dorigo *et al.* (2006) improves on the traditional Ant System of Dorigo & Gambardella (1997) by locally updating the pheromones instead of conducting only one global pheromone update. Once all the ants have built their solutions, the global updating rule is applied to modify the pheromone level on the edges of only the best solutions found so far.

As with decision trees, ACO can be created and built using several techniques. We summarise such techniques and break down the basic ACO structure in the next section The summary is

by no means a comprehensive guideline or list.

4.2.2 Techniques for ant colony optimisation

ACO can be viewed as other constructive metaheuristics such as a population-based algorithm where t solutions are generated at each run. The solutions are governed by a probabilistic constructive mechanism biased towards the numerical pheromones. The pheromone values are adjusted with each run based on the quality of the generated solutions. We can summarise this based on the high-level pseudocode developed by Dorigo *et al.* (2006).

```
1  Algorithm Ant Colony Optimisation is
2  input: problem's construction graphs
3  output: best solution
4
5  function ACO()
6  Initialise
7  while termination condition not met do
8      ConstructSolutions
9      LocalSearch //optional
10     UpdatePheromones
11 end while
12 return best solution
13 end function
```

Listing 4.1: Ant colony optimisation pseudocode

A basic ACO algorithm iterates through three simple functions (López-Ibáñez *et al.*, 2017). The solution is constructed during the ConstructSolutions function and the pheromone values are updated accordingly during the UpdatePheromones function. The current iteration solution can be improved by using a local search function (Mavrovouniotis, Müller & Yang, 2017).

The representation is significantly high-level and ACO can be constructed using several techniques or parameter settings (López-Ibáñez *et al.*, 2017). We investigate the ATM parameters in Chapter 6 as we aim to optimise the solution. We discuss some techniques that can be used within each function to build ACO solutions.

Solution construction: With ACO, each solution is constructed by the virtual organism; each ant represents a probabilistic solution construction function. Parameters can be used to control the significance of the pheromone trail or other heuristic information on the decision probability (Maniezzo, 2002). New solutions are generated with each iteration of the algorithm. López-Ibáñez *et al.* (2017) note that the solution construction function is critical to the development of an ACO solution. The following probabilistic rules can be used to construct ACO solutions:

- Ant System – the probabilistic rule created by Dorigo, Maniezzo & Colorni (1996) could be considered the most popular among ACO solution constructions (Prakasam & Savarimuthu, 2016)

- ANTS – the probabilistic rule was extended by Maniezzo (1999) using his ANTS algorithm combining pheromone trails and heuristic information in an additive function
- Ant Colony System (ACS) – Dorigo & Gambardella (1997) preferred a more deterministic construction function with their ACS algorithm
- AntMiner – The goal of the AntMiner by Parpinelli, Lopes & Freitas (2002*a*) is to extract classification rules from data. Initial experiments noted that the algorithm improves upon the traditional C4.5 algorithm
- cAntMiner – Otero, Freitas & Johnson (2008) extended the AntMiner algorithm with their cAntMiner which uses an entropy-based discretisation method in order to cope with continuous attributes during the rule construction process
- AntMiner$_{ma}$ – the algorithm extends the AntMiner using an archive-based pheromone model to handle mixed attribute types within a data set (Helal & Otero, 2017)

López-Ibáñez *et al.* (2017) mention that to improve efficiency, a "lookahead" can be used that considers multiple components when selecting the best current partial solution. In order to improve efficiency, the number of solutions can be reduced by using candidate lists containing the most promising solution components. The constructed solution greatly depends on the pheromone, which requires a more detailed consideration of the global pheromone update.

Global pheromone update: From the solution construction function, we can deduce that the pheromone heuristic is altered to control the bias during the construction of new solutions (Otero *et al.*, 2012). The pheromone component can be updated globally and locally depending on the algorithm in question. As with normal ants, the pheromones need to evaporate and be deposited throughout the process (López-Ibáñez *et al.*, 2017).

- Evaporation – reduces the pheromone value by some factor with the ultimate goal to relieve the influence of previous pheromone depositions. Pheromone evaporation causes poor decisions to have less influence within the solution construction (López-Ibáñez *et al.*, 2017)
- Deposition – increases the amount of pheromone for a few selected solution components; this has the inverse effect of evaporation on the solution construction

These steps are performed to bias the pheromone trail to ensure that only high-quality solutions are constructed (López-Ibáñez *et al.*, 2017). The pheromones can be initialised by either a small (Alonso, Cordon, De Viana & Herrera, 2004) or a large (Stützle & Hoos, 2000) initial pheromone deposit within the global pheromone update.

The latter would result in a more explorative search phase, as a smaller deposit will reach the best solution much faster. In most cases the pheromone deposit is left to the user to specify. Pheromone updating during the solution construction is referred to as local pheromone updates (Dorigo & Gambardella, 1997). The solution pheromone, chosen by an ant, is updated locally. Stützle & Hoos (2000) note that resetting the pheromone values to their original values contributes to increased exploration within the search space, given enough

time. Often referred to as 'restart', the process resets the pheromone values without altering the global-best solution found. This method of resetting the pheromone values has been quite popular when applied to the travelling salesman problem and the quadratic assignment problem (López-Ibáñez *et al.*, 2017). In both cases a strong local search algorithm was used.

Local search: Commences from a predefined solution and iteratively applies meagre changes, defined by a neighbourhood operator. The local search technique has been used in several ACO solutions by Dorigo & Gambardella (1997), Stützle & Hoos (2000), Gambardella, Montemanni & Weyland (2012) and Mavrovouniotis *et al.* (2017). The local search algorithm can be applied in two ways:

- Best-improvement will replace the current solution with the best
- First-improvement will replace the current solution with the first improving solution in the neighbourhood

Local search algorithms will stop once the entire neighbourhood has been examined and no improved solution was found. When the local search algorithm is combined with ACO, the local search is performed against the solutions constructed by ants.

Based on the literature review, we can summarise the advantages and disadvantages of ACO as follows:

Advantages:

1. Randall & Lewis (2002) investigated several parallelism techniques for ACO. Within ACO, each ant and will start the induction process parallel to one another. This default parallelism allows ACO to be applied in the modern computing environment using parallel processing (Cecilia, Garcia, Ujaldon, Nisbet & Amos, 2011).
2. The discovery of solutions is controlled by the amount of pheromone initiated (López-Ibáñez *et al.*, 2017). To force the quick discovery of good solutions, a small amount of pheromone can be initiated or the evaporation rate can be increased.

Disadvantages:

1. ACO is difficult to analyse theoretically (López-Ibáñez *et al.*, 2017).
2. The virtual organisms consist of a sequence of randomness, as they implement a randomised construction heuristic (Dorigo *et al.*, 2006).
3. The ACO algorithm is dependent on probability distributions that change after each interaction (López-Ibáñez *et al.*, 2017).
4. Due to the nature of ACO to work collectively with no direct communication, the lack of a centralised processor to guide the virtual organisms towards building good solutions is also concerning as it leads to stagnation behaviour (Mavrovouniotis & Yang, 2010).

Despite the complexity, ACO has been implemented within the intrusion detection domain as discussed in the section to follow.

4.2.3 Related work within intrusion detection

Ramos & Abraham (2005) introduced the ANTIDS, an ant colony-based clustering technique to detect intrusions, obtaining an average 90% accuracy on their test data set. The approach offered favourable results, however fell short in comparison with decision trees to detect normal traffic and U2R.

Tsang & Kwong (2006) improved the ant mining cluster algorithm and applied it to the KDD Cup '99 data set for intrusion detection. Their research however used ten-fold cross-validation and is therefore not comparable with research using actual test data sets as they achieve a relatively low error rate.

An ACO algorithm was boosted by Soroush, Abadeh & Habibi (2006) for computer intrusion detection. Their approach improved on the original ant-miner algorithm by partitioning the data set, converting the heuristic function and changing the pruning technique. The research does not make use of a test data set and the detection rate of over 99% is incomparable and can be considered unrealistic.

Wu & Banzhaf (2010) note that recent experiment results for ACO within intrusion detection show that the approach can achieve equivalent or improved performance on traditional methods. The ability of ACO algorithms to function adaptive, parallel and cost-efficient is certainly alluring for intrusion detection.

Kolias, Kambourakis & Maragoudakis (2011) performed a survey on swarm intelligence within intrusion detection. By comparing several experiments using the KDD Cup '99 (or KDD'99 in short) data set, the researchers noted that using swarm intelligence significantly boosts the performance of all the machine learning techniques in which it was applied (Kolias *et al.*, 2011). They concluded that more research following standard complexity analysis alongside intrusion detection metrics is required. We can directly relate that the use of repeatable research methodology and test data sets are required (refer to Chapter 6).

The research performed by Shrivastava & Richariya (2012) combines ACO with a Naïve Bayes algorithm to classify intrusion data. The KDD'99 data set is used for training without a test data set. Their NB-ACO algorithm achieved over 97% detection rates.

Aghdam & Kabiri (2016) created a feature selection process using ACO in intrusion detection, exhibiting very low computational complexity when using a simplified feature set. The KDD'99 data set and the NSL-KDD data sets were used in this research project. As a result, their proposed method reduced the data set features by approximately 90% and the detection errors by approximately 20%.

A recent advance, combining ACO and decision trees by Otero *et al.* (2012), introduced a new way to building decision trees. Their initial experiments with other machine learning data sets showed improvement from the traditional techniques used to build decision trees. We further investigate the ATM algorithm created by Otero *et al.* (2012) in the following chapter.

CHAPTER 5

ANT TREE MINER CLASSIFIER

We introduce the ATM classifier in Chapter 5. Section 1 is an introduction to the work done by Otero *et al.* (2012). Section 2 covers the essential components that make the ATM unique from the algorithms discussed in Chapter 4. In the last section we discuss related work within other domains.

5.1 Introduction

The unknown spectrum of inducing decision trees with ACO algorithms has been a new research topic since 2012 (Otero *et al.*, 2012). The divide-and-conquer principle is used to induce a decision tree: it involves iterating from the top down, selecting the best attribute to label an internal node of the tree. However, this process becomes moot as the appropriate attribute has to be selected based on a heuristic evaluation.

In the previous chapter, decision trees were discussed as reported in the related literature. Otero *et al.* (2012) propose a new method to induce decision trees – the method follows the traditional structure of ACO. The proposed ATM follows the same structure as ACO. The following pseudocode summarises the Ant Tree Miner algorithm as created by Otero *et al.* (2012).

```
1  Algorithm Ant Tree Miner is
2      input: training examples, list of predictor attributes
3      output: best tree identified
4
5  InitialisePheromones
6  ComputeHeuristicInformation
7  treeGB is emptySet
8  m is 0
9  while m < maximum iterations  and not CheckConvergence do
10     treeIB is emptySet
11     for n is 1 to colony_size do
12         treeN is CreateTree(Examples, Attributes, -)
13         Prune(treeN)
14         if Q(treeN) > Q(treeIB) then
15             treeIB is treeN
16         end if
17     end for
18     UpdatePheromones(treeIB)
19     if Q(treeIB) > Q(treeGB) then
20         treeGB is treeIB
21     end if
```

```
22        m is m + 1
23    end while
24    return treeGB
```

Listing 5.1: Ant tree miner pseudocode by Otero et al. (2012)

The heuristic information for each attribute is first calculated, and then each ant (during the while loop) in the colony creates a new decision tree until the maximum number of iterations is reached or the algorithm converges.

The virtual ants create each decision tree using high reliance on the pheromone and heuristic information. To avoid overfitting, a decision tree is pruned once the tree is created. The decision tree is evaluated (treeIB) against the current iteration's tree. The latter is only updated if the quality is better and the tree is then used to update the pheromone values. A global-best tree (treeGB) is tracked, and a new iteration of the algorithms starts. Upon conclusion, the global-best tree is returned as the discovered decision tree.

The ATM classifier differs significantly from the AntMiner algorithm through building decision trees instead of rules. We discuss the components of the ATM in the section to follow.

5.2 Components of Ant Tree Miner

In this section we discuss the numerous components that form the ATM algorithm and once trained, a classifier. Some of these components e.g. pruning and pheromone have already been introduced in the previous chapters.

5.2.1 Construction graph

Each virtual ant constructs candidate solutions by traversing the construction graph. The dynamic construction graph consists of N vertices that represent the attributes within the data set, one vertex per attribute (Otero *et al.*, 2012). Edges coinciding with different conditions related to values from the domain of attributes connect each vertex. Within the ATM algorithm, each ant would start at the virtual *'start'* vertex, considering that N edges connect the start node to every x_i attribute vertex of the construction graph.

The construction graph's interpretation of nominal and continuous attributes are summarised as follows:

1. **Nominal attributes:** Edges represent the condition where the attribute x_i has the value v_ij (Otero *et al.*, 2012). However, ants cannot select the same nominal attribute multiple times in the same path of the tree.
2. **Continuous attributes:** A dynamic discretisation procedure is followed to create discrete intervals for continuous values. This is due to continuous attributes not featuring any predefined set of fixed intervals. ATM uses two discretisation techniques, one similar to C4.5 selecting threshold values according to the information gain. The second technique is based on the minimum description length (MDL) as proposed by Fayyad & Irani (1993).

Based on the nature of discretisation, the construction graph is dynamic and adapts based on the current path followed by an ant. The process is completed for the subset of training examples used in the discretisation procedure (Otero *et al.*, 2012). Figure 5.1 illustrates a construction graph based on the decision tree example from the previous chapter. In the example, the construction graph is composed of three vertices namely 1. {TCP, UDP}, 2. {L=0, L=1} and 3. {s<3,s>3}. The start can be considered a virtual *'start'* vertex.

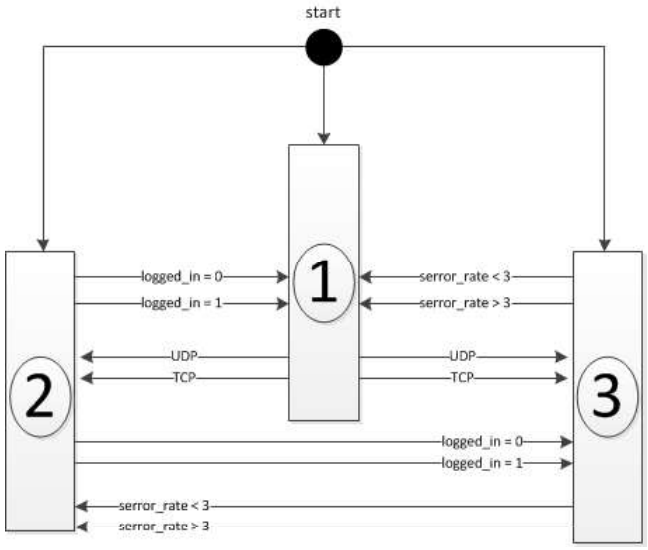

Figure 5.1: Construction graph example

5.2.2 Heuristics

To understand the heuristics used within ATM, we first need to unpack the C4.5 induction technique by Quinlan (1993). For the application of C4.5 within decision trees, refer to the previous chapter. The C4.5 uses an entropy-based criterion to select the best attributes; in essence, the entropy measures the impurity of a set of examples relative to their class attributes (Quinlan, 1993). The entropy is used to calculate the information gain of an attribute A and should correspond to the reduction in entropy achieved by splitting the training examples into V subsets, where V is equal to the total distinct values in the domain. We can define information gain as denoted in the formula below:

$$InfoGain(S, A) = Entropy(S) - \sum_{j=1}^{v} \frac{|S_j|}{|S|} \cdot Entropy(S_j) \tag{5.1}$$

where $|S_j|$ is the total number of samples in the subset of S. The information gain penalises attributes that divide the training into very small subsets, noted as split information:

$$SplitInfo(S, A) = -\sum_{j=1}^{v} \frac{|S_j|}{|S|} \times \log_2(\frac{|S_j|}{|S|}) \tag{5.2}$$

The penalty is required due to the dividing attributes' high information gain within the small subset. The information gain of an attribute A is calculated from the InfoGain and SplitInfo

measure as denoted below:

$$GainRatio(S, A) = \frac{InfoGain(S, A)}{SplitInfo(S, A)} \tag{5.3}$$

The heuristic information for each attribute vertex x_i is based on the estimated quality in relation to improving the predictive accuracy of the underlying decision tree. The ATM uses the same heuristic information as with the C4.5 algorithm and can be denoted as:

$$n_{xi} = GainRatio(S, x_i) \tag{5.4}$$

where x_i corresponds to the i-th attribute vertex and S corresponds to the set of training examples. The GainRatio is described in formula 5.3, where a continuous attribute's information gain is calculated by dynamically selecting threshold values that define discrete intervals and ensuring that the training example is split into subsets. Once completed, the information gain is calculated with the assumption that every discrete interval serves a dissimilar value and therefore a different subset of the training set. The ATM differs from the traditional C4.5 through selecting attributes based on the information gain and pheromone values (Otero *et al.*, 2012). Otero *et al.* (2012) note that the pheromone feedback improves the quality solution by considering the entire decision tree (global attribute evaluation), consequently compensating for imprecisions and greedy information gain ratio measures.

5.2.3 Solution construction

With the heuristics out of the way, we delve into the solution construction process. The process used by the ATM is similar to that of the ACO algorithm. The classical divide-and-conquer approach is followed to construct candidate decision trees. The ATM differs however from traditional techniques by stochastically selecting attributes based on the pheromone and heuristic information (Otero *et al.*, 2012). The virtual ants apply the probabilistic rule with each iteration to determine which attribute vertex should be visited. Otero *et al.* (2012) denote the probability of an ant, p_i to visit the vertex x_i with the following formula:

$$p_i = \frac{\tau(E, L, x_i) \cdot \eta_i}{\sum_{i \in \mathcal{F}} \tau(E, L, x_i) \cdot \eta_i}, \forall i \in \mathcal{F}, \tag{5.5}$$

Refer to the paper written by the authors for further information on the formula and the application thereof (Otero *et al.*, 2012).

```
1  Algorithm Ant Tree Miner Solution Construction is
2      input: training examples (Examples),
3             list of predictor attributes (Attributes),
4             current edge (Edge)
5      output: root node of the decision tree
6
7  A ← probabilistically selects an attribute from Attributes to visit given the
       current Edge
8  root ← creates a new decision node representing attribute A
9  conditions ← ?
10 if A is a nominal attribute then
11     Attributes← Attributes − {A}
12     for all value v_i in domain of A do
```

```
13          conditions ← conditions + {A = v_i}
14      end for
15  else
16      conditions ← Discretise(A, Examples)
17  end if
18
19  for all attribute condition T in conditions do
20      branch_i ← new branch representing T of root
21      subset_i ← subset of Examples that satisfies T
22      if subset_i is empty then
23          Add leaf node with the majority class label of Examples below branch_i
24      else if all examples in subset_i have the same class label then
25          add a leaf node with the class label of subset_i below branch_i
26      else if number of examples in subset_i is below threshold then
27          add a leaf node with the majority class label of subset_i below branch_i
28      else if Attributes is empty then
29          add leaf node with the majority class label of subset_i below branch_i
30      else
31          add the subtree returned by CreateTree(subset_i, Attributes, branch_i) below branch_i
32      end if
33  end for
34  return root
```

Listing 5.2: Ant Tree Miner solution construction pseudocode by Otero et al. (2012)

The pseudocode in figure 5.2 represents the decision tree construction procedure used in the ATM algorithm. The virtual ants originate from the 'start' node and follow the edge '-' to build candidate decision trees, given the full set of training examples and predictor attributes. Each selected attribute is the origin of another decision node.

As mentioned earlier in the chapter, nominal and continuous attributes are treated differently. Branches of the decision nodes are created based on the set of attribute conditions selected. The training set is split into subsets e.g. examples for each attribute condition (branch). The construction procedure evaluates whether a leaf node should be added below the current branch or, instead, recursively add a subtree below the current branch.

The deterministic procedure to add leaf nodes to a candidate decision tree is based on the following conditions (Otero *et al.*, 2012):

1. None of the training examples satisfy the attribute condition represented by the current branch e.g. the current subset of training examples is null.
2. All examples in the current subset are associated with the same class label.
3. The number of examples in the subset is below a user-defined threshold.
4. The set of available predictor attributes is null.

A leaf node representing the class label prediction is only added below the current branch if any of the given conditions are met. If not, the construction procedure is applied recursively (Otero *et al.*, 2012). Once completed, the root node of the candidate tree is returned to conclude the

construction procedure. We discuss the pruning procedure used with ATM in the section to follow.

5.2.4 Pruning

We have already discussed the necessity and techniques used to prune decision trees in Chapter 4. The ATM prunes each candidate decision tree once the construction procedure is completed. The construction procedure builds candidate decision trees until there are no more attributes available or the training set is depleted, or each training example is associated with an exact class label (Otero *et al.*, 2012).

The error-based pruning technique used by the C4.5 algorithm discussed in Chapter 4 is similar to that of the ATM. The ATM prunes decision trees using two simple steps:

- Step one – creates a potentially overfitted model by pruning the tree to fit the training data [training model]
- Step two – prunes the overfitted model to increase generalisation power [generalised model]

The pruned training model is created by replacing decision nodes based on two conditions e.g. its most common used branch or the leaf node that guarantees a higher classification accuracy on the training data. This is a crucial step as it removes decision nodes that would have an adverse effect on the classification accuracy. The step is completed until there are no more benefits for the replacement or the tree consists of only one leaf node.

The generalised model follows the error-based pruning technique noted in Chapter 4. Instead of replacing leaf nodes that could potentially improve accuracy, it replaces leaf nodes that lead to a lower estimated error rate.

A decision model based on the training data could be unable to generalise when exposed to test data, leading to a classic low detection rate with overfitted decision models. Overfitted models based on training data were discussed in the previous chapter. We make note of this again in Chapter 6, as we discuss the model built by the ATM classifier on the poor training data for U2R and R2L attacks. The contrast in tactics in step two is necessary to ensure that the best build decision model is produced. The process concludes the pruning of the candidate decision trees and we discuss the updating of pheromones and the pheromone matrix in the next section.

5.2.5 Pheromones

The crucial part of the ATM algorithm is the updating and representation of pheromones. To keep track of the pheromone values, a pheromone matrix is used. The virtual ants travel in the construction graph as each edge represents an attribute condition. Therefore each choice made by the virtual ants can be expressed as an edge that is followed to create a decision tree, and each entry in the matrix is represented by a triple [$edge_{ij}$, $level$, x_k] where $edge_{ij}$ is the corresponding edge of the j-th attribute (Otero *et al.*, 2012). The level within the decision tree where $edge_{ij}$ appears is noted as the $level$ and the destination denoted as x_k. According

to Otero *et al.* (2012), edges that lead directly to a leaf node representing a corresponding prediction are not represented in the pheromone matrix as they are either deterministically introduced or during the tree creation process, or by the pruning process.

The process of pheromone updating has been introduced in the previous chapter. The ATM algorithm uses the popular MAX-MIN technique created by Stutzle & Hoos (1998). The pheromone values are limited to the intervals set for maximum and minimum, which are updated after each new global-best solution. For a detailed explanation, refer to the paper by Stutzle & Hoos (1998).

The pheromone values are updated in two simple steps:

1. Step one – using the pheromone evaporation rate (further discussed in Chapter 6), the pheromone value for each entry in the matrix is decreased by a factor of ρ as defined by the user.
2. Step two – based on the quality of the candidate solution, the pheromone value for the entries related to each branch used for the iteration-best candidate solution is increased. The quality is measured based on the following formula (Otero *et al.*, 2012):

$$Q = \frac{N - Error}{N} \tag{5.6}$$

   where N represents the number of training examples and $Error$ the estimated classification errors of the candidate tree; the lowest error rate provides the highest quality solution.

Once these steps are completed, the pheromone update process starts. Otero *et al.* (2012) created the following pheromone update rule as described in the formula below:

$$\tau_{(E,L,x_i)} = \begin{cases} \rho \cdot \tau_{(E,L,x_i)}, & \text{if } (E, L, x_i) \notin tree_{ib}; \\ \rho \cdot \tau_{(E,L,x_i)} + Q(tree_{ib}), & \text{if} (E, L, x_i) \in tree_{ib}; \end{cases} \tag{5.7}$$

Within the formula, ρ represents the MAX-MIN evaporation factor, $\tau_{(E,L,x_i)}$ the pheromone value associated with the entry (E, L, x_i), E the attribute condition of the corresponding edge, L the level, x_i the edge's destination and the iteration-best decision tree noted as $tree_{ib}$.

The updating of pheromones concludes the components of the ATM algorithm. In the next section we discuss the initial paper and related work based on ATM.

5.3 Related work within other fields

It is important to note that in this research, it is the first time that decision trees induced by ant colony optimisation is implemented within the intrusion detection domain. This argument is supported by the related work for the ATM illustrated in the summary below. The related work in the fields refer to the induction of decision trees using ant colony optimisation.

A timeline table summarising the recent developments related and leading to the development of the ATM was constructed.

Table 5.1: Related research timeline

2010	Boryczka & Kozak (2010) introduce the ACDT algorithm creating binary decision trees.
2011	The ACDT algorithm is adapted for continuous attributes by Boryczka & Kozak (2011).
2012	Boryczka & Kozak (2012) create the Ant Colony Decision Forest (ACDF) algorithm, an ensemble extension of ACDT.
2012	Otero *et al.* (2012) create the ATM algorithm, not limited by creating only binary decision trees, and evaluate solutions based on pheromone values and the information gain.
2014	Kozak & Boryczka (2015) extend the ACDF algorithm by applying the ensemble boosting technique.
2014	Salama & Otero (2014) extend the ATM algorithm to build a new multi-tree classification model.
2014	The ATM is extended with the bagging ensemble technique by Chennupati (2014).
2015	The ACDT algorithm is extended as the Novel Numerical Ant Colony Decision Tree (nACDT) algorithm by Surjandari *et al.* (2015) for estimating the duration of drydocking.
2015	Evaluation techniques used within the ATM are investigated by Salama, Abdelbar & Otero (2016).
2016	Kozak & Boryczka (2016) further investigate the use of heuristics and a pheromone trail to build decision trees, similar to the approach used in ATM.
2016	Boryczka *et al.* (2016) use the ACDT to classify e-mails.

It is clear that the combination of decision trees and ant colony optimisation have received some attention since its rise early in 2010; however, implementing such algorithms within other domains has been extremely limited, as we could only note the research performed by Boryczka *et al.* (2016) to categorise e-mails and Surjandari *et al.* (2015) to estimate the duration of dry docking as examples.

ATM differs significantly from other research using the traditional Ant-Miner created by Parpinelli, Lopes & Freitas (2002*b*), instead of building a set of rules for the ATM build decision trees. It can be noted that some of the research contributed significantly to the development and understanding of the ATM algorithm, therefore these are discussed in more detail below.

The ACDT by Boryczka & Kozak (2010) can only build binary decision trees and evaluate candidate solutions using the pheromone level and heuristic information. The heuristic

information is based on the Twoing criteria used in the CART algorithm.

The ATM uses the information gain similar to that of the C4.5 algorithm, and based on the research performed by Otero *et al.* (2012) it significantly outperforms the ACDT and variations. Otero *et al.* (2012) note that the predictive accuracy of the ATM is on average statistically significant better than C4.5, CART and ACDT.

The ATM scores only slightly lower than the best algorithm for the size of the classification model, e.g. the number of leaf nodes. The CART model is known for its ability to build models with a smaller classification model. Regrettably, this greatly impacts the predictive accuracy (Otero *et al.*, 2012).

This is clearly not the case with the ATM as it can build smaller classification models to its counterparts while retaining superb predictive accuracy. It is important to note that the ATM algorithm's computational performance is significantly lower than the other algorithms, as noted by (Otero *et al.*, 2012). The effect is reduced due to the ATM's ability to operate in a cluster. Utilising this functionality will significantly increase the runtime speed. The ATM's slow performance does not impact its practicality for intrusion detection. The ATM algorithm will be better suited for offline intrusion detection systems. Offline intrusion detection systems capture live network traffic but only analyses it as historical data (SunilKumar, 2012).

The ensemble ATM algorithm created by Chennupati (2014) focuses on the instability within the ATM classifier and concludes with a more stable algorithm through the use of ensemble techniques. We discuss the potential for further research in Chapter 8.

From the literature review, we can understand the need to implement the algorithm within other domains. In Chapter 7 we discuss the methodology and processes used to implement the ATM algorithm within the intrusion detection domain.

CHAPTER 6

EXPERIMENTAL METHODOLOGY AND RESULTS

6.1 Introduction

For this research, a pragmatist philosophical stance was adopted, as the researcher focused on 'what works' as the truth and did not participate in the choice associated with the opposing truth and reality. Deductive reasoning was applied throughout the study. The primary goal of the study was to create and contribute theory with regard to the ATM classifier within the intrusion detection domain as well as the machine learning domain. The strategic approach to the research study has been identified as experimental; the researcher experimented with the Ant Tree Miner classifier on intrusion detection data. The research mostly used quantitative analysis and the choice for analysis could be considered a mono method. The results generated have been statistically analysed using the performance derivatives Receiver Operating Characteristic (ROC) analysis and cost evaluation, further discussed in section 6.5. Cross-sectional research design was adopted, as quantitative analysis measures have been used on intrusion detection data sets to measure the performance and accuracy of the classifier at a given period.

The WEKA toolkit created by Russell & Markov (2006) was used to perform data analysis and the pre-processing of data. The data set created by Tavallaee *et al.* (2009) improves the issues associated with the famous KDD'99 data set used in several research projects. The researcher elected to use the NSL-KDD data set for this research project. The motivation and discussion of these data sets is done in section 6.2 of the thesis.

The researcher answered several research questions based on the literature review and experiments performed. The literature provided a much needed theoretical background and a baseline for decision trees and swarm intelligence research previously performed specifically in the intrusion detection field. The goal of this approach was to evaluate and compare the mentioned key components to form a holistic overview of the machine learning techniques used for intrusion detection and how the proposed classifier compares.

The researcher used the Experiment Life Cycle (ELC) of the Deter project proposed by Mirkovic, Benzel, Faber, Braden, Wroclawski & Schwab (2010) to manage the experiments. Diagram 6.1 illustrates the ELC used with this research project.

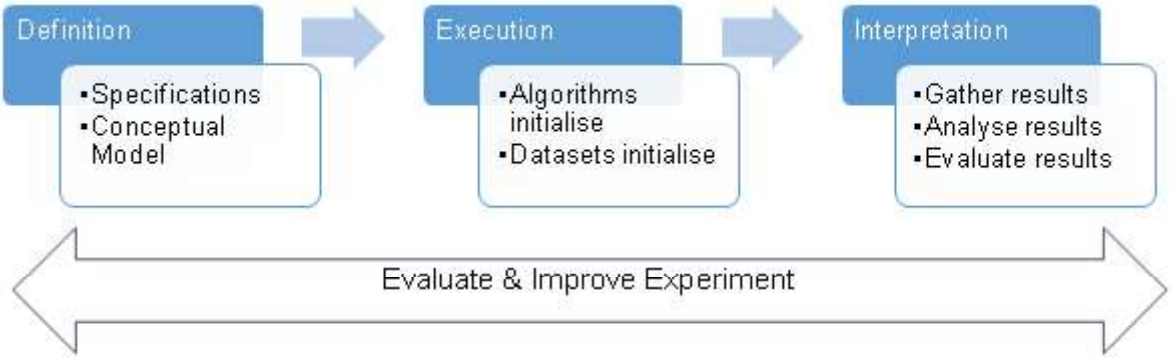

Figure 6.1: Proposed ELC

The experimentation process was carefully designed to improve on previous research, especially using machine learning within the intrusion detection domain. The conceptual experimental process can be outlined as follows:

1. Prepare data sets.
2. Feature selection.
3. Classifier initialisation.
4. Experiment run.
5. Results gathering.
6. Results evaluation.

1. Prepare data sets: The preparation of the data sets is a crucial part of the experiment and the foundation on which the research is built. To allow the experiments to be repeated, the process is kept transparent and all data sets used are shared. Although the NSL-KDD data set has been chosen as the main data set for experimentation, a pre-processing and validation process was performed. The NSL-KDD data set has been made available publicly by Tavallaee *et al.* (2009) and Tavallaee, Bagheri, Lu & Ghorbani (2011). The data set consists of several data sets used for training and testing classifiers. To align the experiment with the ELC, the following data sets were used:

- Training data set – used to train the classifier
- Validation data set – used to validate the classifier
- Testing data set – included unknown data used to evaluate the classifier

Training data set – classifiers are trained using training data. The researcher modified each NSL-KDD data set to replace the classification field or field 42 with a field, called xAttack. The data set was also pre-processed using feature coding. In other words, categorical feature encoding was used to change the text-based categories to numeric values; therefore, the nominal fields were represented as numeric categories instead of text. Nominal fields represent certain classes or categories e.g. Male or Female, House, Apartment, Duplex, etc. For training, the classifier KDDTrain20% (also referred to as Train20%) data set was used. The

data set and its features are further discussed in section 6.2. The pre-processed data set was split using the xAttack field for each attack category, thereby allowing the classifier to be trained per attack type. Thus, the data set has now become a multi-class data set, since we need to split different attack types instead of attack or normal traffic. The xAttack field is, however, only used to split the data sets and never to classify, and the data are converted back to binary classification after each data set split. The 'per attack' type data set only includes the relevant attack and normal network traffic. Therefore the prediction class only consists of 0 [normal traffic] and 1 [attack traffic]. This totalled the training data sets to five, one for each attack category and one full 20% training (also referred to as Train20%) data set.

Validation data set – on the initial run of the experiment, to validate the classifier with different parameter settings, a validation data set was created by splitting the 20% training data set into 66% training and 34% validation.

Test data set – the same process was performed for the test data set, which is further discussed in section 6.3. The test data set's data were never altered, only the features were reduced for testing. Feature reduction only entails removing features to ensure that both training and test data sets consist of the same features. The feature selection process was only performed on the training set, thereby ensuring reliability and comparability as with the original test data set.

2. Feature selection: Feature selection extracts statistically relevant features from the data sets. For this research project, the information gain and ranker feature selection methods were used. The methodology and the process used for feature selection are discussed in Chapter 3. The process was done using WEKA[1], a machine learning tool created by Hall, Frank, Holmes, Pfahringer, Reutemann & Witten (2009). After the relevant features for each attack type had been established, the five data sets were trimmed to only include those highlighted during the feature selection process. The original NSL-KDD data set has now been pre-processed, split per attack type and trimmed using feature selection. The final experiments were performed on the modified NSL-KDD Train20% data sets.

3. Classifier initialisation: The main purpose of the research study was to experiment with the ATM classifier within intrusion detection. The Java binaries can be obtained from the project Myra Github repository (Otero, 2016). Version 4.1 of the Ant Tree Miner classifier was used for this research project. This version offers significant improvement on the previous 3.7 version, including parallel processing and a compelling performance increase. We further discuss the setup of the classifier in section 6.4. With the first cycle of the experiment run, i.e. the validation cycle, we experimented with different parameters for the ATM classifier.

4. Experiment run: The experiments were performed on a Dell E6320 running OpenSuse 42.2 and operating with an Intel Core i7-2620M processor and 8GB ram. It is important to recognise the experiment cycle as the steps performed from step 1 to 6.

[1] Available for download at `http://www.cs.waikato.ac.nz/ml/weka/`

Two experiment cycles were performed as follows:

- Experiment with split training and validation data set (only full 20% training data set)
- Experiment with final modified training and test data set

We first experimented with the 66 – 34 split training and validation data sets to establish the overall ability of the classifier to accurately classify malicious data. This approach was followed to test different parameters within the classifier and identify which parameters would perform best with intrusion detection data. During this cycle, the feature selected training data set was the only data set used.

The approach to limit the data set to only train the classifier per attack type allows digging deeper into the classifier's abilities. To provide competitive results with previous research performed, we also experimented with the full 20% training data set. The prediction models built on the full 20% training data set can be used to create an ensemble version of the ATM classifier. The ensemble technique and results are discussed in section 6.7.

The split attack type approach also eliminated the issues with the unfair attack representation in the training data set, inherited from the KDD'99 data set. Each experiment was performed 10 times, and the results were averaged for overall ability. The experiment cycle is not uncommon, as fellow researchers have used the same approach Ramos & Abraham (2005) and Geramiraz, Memaripour & Abbaspour (2012). In their papers, the researchers identified key features to classify the four attack classes. The first steps for the experiments were to assess the features selected and the classifier's parameters before a test is done against a test data set. To further knowledge on the classifier, the second cycle included experiments with and without feature selection enabled, and this added another dimension to the results. The experiments were performed with the assistance of the CSIR Cybersecurity Centre of Innovation.

5. Results gathering: The classifier's results were collected for each experimental cycle. A sample output from the ATM classifier is shown in figure 6.2. The result file for each experiment was stored for later analysis. The analysis of each experiment was performed with Microsoft Excel.

6. Results evaluation: The classifier's initial ability to classify the intrusion detection data was validated in the first experiment cycle. During the first cycle, we evaluated the results based on the classifier's ability to classify the validation data only. We experimented with different parameters during the initial cycle to identify the best performance. Results were evaluated as per the guidelines outlined in section 6.5.

In order to evaluate the classifier, a specific test data set was used for the final experiment cycle. The techniques, based on an extensive literature review, provided the researcher with the tools to analyse the ATM's performance. The results were compared with other tree-based and ant colony optimisation techniques performed on the NSL-KDD data set.

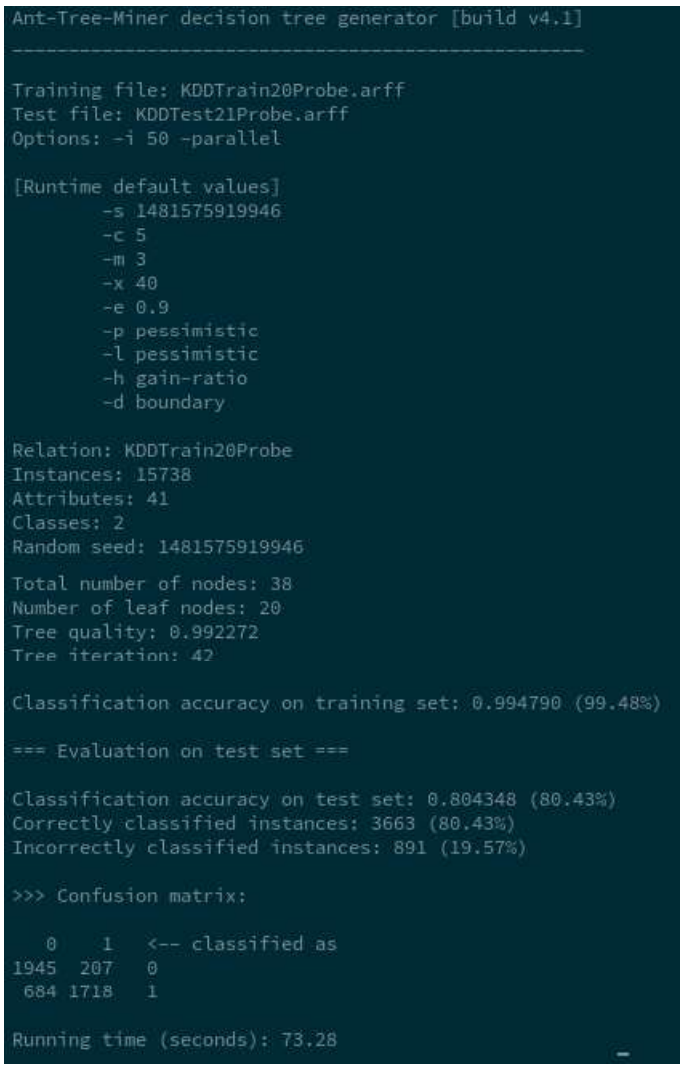

Figure 6.2: ATM classifier

6.2 Data sets

In this section, we discuss and motivate the NSL-KDD data set as the selected training data set and test data set. First, we analyse the KDD'99 data set and highlight critical research studies involving the data set. Any research using the KDD'99 data set has been selected as flawed and disregarded in this research project. The motivation thereof is done in the discussion to follow.

The next subsection introduces the NSL-KDD data set. Emphasis is placed on the data set structure, features, attack platforms and research projects evaluating the data set. It is crucial to understand that data sets are used to train the classifier and test its effectiveness. We therefore use the data sets for evaluation of intrusion detection systems. To create effective detection models, the classifier requires a vast amount of labelled data. Labelled network data is significantly hard to come by due to the time constraints, legal considerations and complexity of labelling each record.

One can imagine the sheer amount of time required to build such data sets. Thus, the activity becomes time-consuming and a costly effort. Bhuyan, Bhattacharyya & Kalita (2015)

investigated the importance of data sets within intrusion detection research. The following reasons justify using data sets for intrusion detection:

1. **Repeatability of experiments:** A critical part of experimentation is to recreate results and verify statements by other researchers using the same approaches (Bhuyan *et al.*, 2015).
2. **Validation of new approaches:** New research projects drive new methods, and classifiers should be continuously developed. It is necessary that every new approach be validated (Bhuyan *et al.*, 2015).
3. **Comparison of different approaches:** New methods and classifiers not only require extensive validation, the improvements also need to be quantifiable. Using the same data sets allows for comparative results and stability whenever black box approaches are used (Bhuyan *et al.*, 2015).
4. **Number of features:** The optimal set of attributes or features should represent normal and malicious attack instances (Bhuyan *et al.*, 2015).

The research by Bhuyan *et al.* (2015) identified the following requirements for data sets to be used in intrusion detection:

- Real world – data sets should represent real world scenarios and realistic network traffic
- Completeness in labelling – the labelling of traffic as malicious must be backed by proper evidence for each instance. Network traffic can be modelled using two techniques: packet-level and flow-level (Venkataramanan, Jeong & Balaji, 2011). The latter is less accurate than its counterpart. The ideal labelled data set should provide packet and traffic flow levels for each malicious activity
- Correctness in labelling – the labelling for each network traffic instance must be correct
- Unbiased – the data set must be unbiased in terms of size for attack types and network traffic

The requirements for data sets within intrusion detection have been further supported by Tavallaee *et al.* (2009) who criticised the KDD'99 data set. A summary of several alternative intrusion detection data sets is provided in Appendix D. A shorter version of the results is depicted in the data set summary table 6.1.

Table 6.1: Data set summary

Data set	Availability	Recent citations
DARPA 98/99	Y	62
NSL-KDD	Y	424
MAWI WORKING GROUP	Y	52
CAIDA	P	2
PREDICT/IMPACT	P	Not available
UNSW-NB15	Y	74

The values: Y denotes YES, P indicates PARTIAL. The table aims to highlight the differences

between data sets and motivate the selection of the NSL-KDD data set. The Google Scholar count column outlines the number of citations for a given data set between 01/01/2017 and 24/09/2018. Based on the result, it is clear that the NSL-KDD data set is preferred by other researchers as an intrusion detection data set. Appendix D provides a detailed outline on how the table results were obtained. The main purpose of the next sections is to analyse the NSL-KDD data set and provide insights into the limitations of its predecessor, the KDD'99 data set.

6.2.1 KDD'99 data set

Sommer & Paxson (2010) highlight the importance of using real-network data sets and the need for improved data sets on the old KDD'99 data set, a very popular choice within the research community. The comparison and evaluation of the different data sets can almost become as convoluted as the benefits and drawbacks of each machine learning method.

Intrusion detection systems will have to measure against benchmark data sets to accurately verify the effectiveness and accuracy of the system. However, these benchmark data sets are synthetic in nature and do not represent real-world network environments. The statement is further supported by Shiravi, Shiravi, Tavallaee & Ghorbani (2012) and reiterates how popular the KDD'99 data set is, despite the vast amount of criticism.

A survey study done by Ahmad *et al.* (2009) reveals the popularity of the KDD'99 data set. Shiravi *et al.* (2012) further criticise the KDD'99 data set, highlighting that network behaviour and patterns change and so does network intrusion, and we require a more dynamically generated data set. Tavallaee *et al.* (2010) further note in their survey study in which they reviewed 276 research articles based on anomaly intrusion detection systems that only 7% of the studies test the robustness of their system through proper test data sets, representing real world networks. This is a rather alarming number, considering we aim to solve real world problems with the research.

One can draw from the research that within a simulated environment the model needs to be trained using the most real world data set possible. It is clear that the data set and evaluation of the systems play a critical role in the deployment of the system; one can predict that the low rate of real network testing contributes to the slow adoption of production-based intelligent intrusion detection systems.

Innovating the intrusion detection research area, Defence Advanced Research Project Agency, US (DARPA) set out to generate a much-needed intrusion detection audit data set. The data set was set to be shared among researchers, evaluate intrusion detection systems, including a wide variety of attacks, and measure both attack detection rates and false alarm rates for realistic normal traffic. To overcome legal issues with the publication of confidential information or mitigate causing disruption within real network environments, an extensive test bed was setup at MIT's Lincoln Laboratories (Lippmann, Haines, Fried, Korba & Das, 2000). The environment simulated the operational networks for a typical US Air Force LAN over a period of two months.

The famous Knowledge Discovery and Data Mining (KDD) Cup competition in 1999 focused on intrusion detection. The KDD'99 data set was created by Lee & Stolfo (2000). The data set consists of the 1998 DARPA data set, which has been pre-processed. Features within the data set were split into the following categories:

- Basic features
- Content features
- Traffic features

To summarise and outline the KDD Cup (2016) data set, several tables and visualisations were created. It is quite common to use the 10% version of the data set for training classifiers as this constitutes a smaller sample of the full data set. Table 6.2 shows the number of attacks per category within the KDD'99 10% data set.

Table 6.2: Attack and category summary for KDD'99 10% data set

Attack	Count of attacks	Attack category
back	2203	DOS
land	21	DOS
neptune	107201	DOS
pod	264	DOS
smurf	280790	DOS
teardrop	979	DOS
normal	97277	Normal
ipsweep	1247	Probe
nmap	231	Probe
portsweep	1040	Probe
satan	1589	Probe
frtp write	8	R2L
guess passwd	53	R2L
inmap	12	R2L
multihop	7	R2L
phf	4	R2L
spy	2	R2L
warezclient	1020	R2L
warezmaster	20	R2L
buffer overflow	30	U2R
loadmodule	9	U2R
perl	3	U2R
rootkit	10	U2R

A critical research study performed by Tavallaee *et al.* (2009) highlights several serious issues with the KDD'99 data set. Another concern with the KDD'99 data set is the number of redundant records (Tavallaee *et al.*, 2009). Due to the significant number of redundant records, learning classifiers become biased towards the more frequent records. This is the primary reason why any results using the KDD'99 data set were not included in this research study.

To highlight the unfair attack representation, the following chart was created:

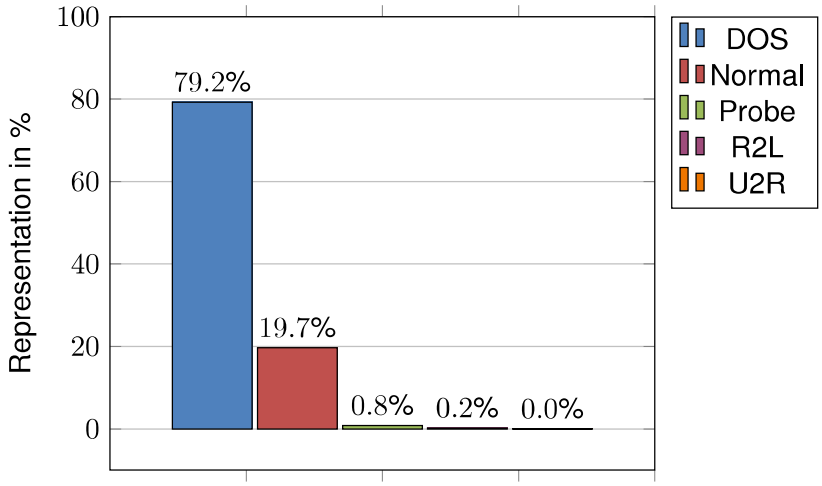

Figure 6.3: Attack representation within KDD'99 10% data set

We can clearly see that the data set is biased towards the DOS attacks, with very little representation for the R2L and U2R attacks. The very few U2R attacks almost represent 0% of the total data set. This is however unrealistic as experience indicates that the popularity of probe attacks are the most commonly used (McHugh, 2000; Vasilomanolakis *et al.*, 2016).

Portnoy, Eskin & Stolfo (2001) split the KDD'99 data set into subsets of ten or 10% of the data. The uneven distribution made cross-validation tests difficult to perform on the data sets. Portnoy *et al.* (2001) mark the variance in attack categories as unrealistic, leading to flawed evaluation. The basic features within the data set represent typical TCP/IP connections.

In general, these features lead to a delayed detection. Table 6.3 represents the basic features within the KDD'99 10% data set.

Table 6.3: Basic features within KDD'99 10% data set

Attribute name	Description	Attribute type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g. http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	if from same host/port =1; 0 if not	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Figure 6.4 below visualises the traffic per protocol within the data set as either normal or attack data.

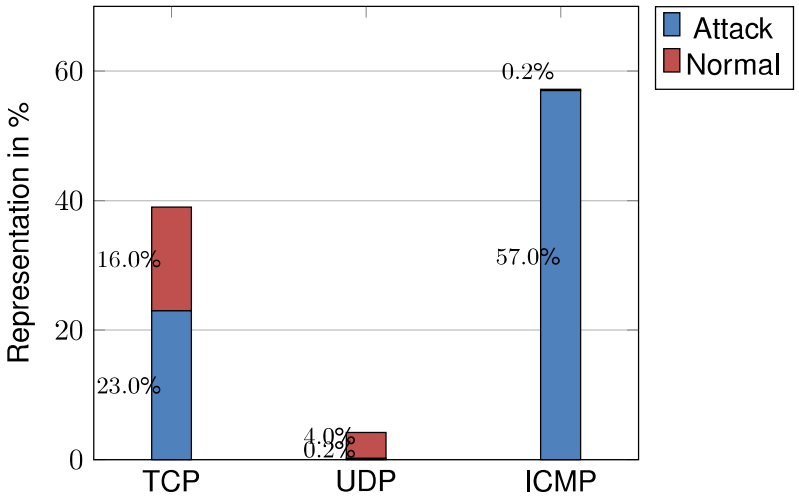

Figure 6.4: Attack representation per protocol within KDD'99 10% data set

The most favoured protocol is the ICMP. Generally, one can assume it is a connectionless protocol, favoured for diagnostics. For example, querying an UDP protocol which is closed, would return an ICMP "port closed" message. However, the ICMP protocol has been heavily favoured within the data set above the others by the creators.

Specific attacks have been discussed in detail in Chapter 2. It is crucial to once again touch on the topic to fully comprehend the issues within the KDD'99 data set. It can be further argued that UDP ports should have much better representation. TCP ports are designed to guarantee that packets are delivered. However, UDP ports do not function this way, thus displaying much more vulnerability than TCP ports. UDP ports represent the smallest subset of the data set when we observe the representation in figure 6.4.

Correa, Nixon & Bienkowski (2016) mention that UDP protocols are the most commonly abused with DDOS attacks. The protocol is common in Microsoft-based products, especially SQL Server. UDP ports are favoured for a myriad of reasons such as speed, stateless nature and small size. UDP has also become popular with services such as DNS, Dynamic Host Configuration Protocol (DHCP), and Voice over Internet Protocol (VOIP) (Zhu, Hu, Heidemann, Wessels, Mankin & Somaiya, 2015; Rajput, Tewani & Dubey, 2016; Abdelrahman, Saeed & Alsaqour, 2016).

To further the perspective, Akamai (2016) reveals in their recent Security Report for Q3 2016 that DDOS attacks have increased by 75% since Q3 2015. The report also shows that the most used DoS attack vectors are, in fact, UDP-based by 25%, miles ahead of the other attack vectors. The fact that UDP ports do not maintain connection states and begin sending data immediately without any start-up overhead should warrant better representation. These characteristics allow the UDP protocol to handle more clients than other protocols (Lee, Carpenter & Brownlee, 2010).

The small UDP header increases transfer speeds significantly above those of TCP protocols. The send rate is not controlled, allowing attackers to send data even if the connection becomes congested. The combination of factors mentioned above makes UDP ports more dangerous than TCP protocols. TCP ports might be more popular, but a representation of only 0.2% attack traffic for UDP ports are not justified at all. To detect the attacks using intrusion detection, more features are required that focus on suspicious behaviour within the packet data. These features are referred to as content features and represented in table 6.4.

Table 6.4: Content features within KDD'99 10% data set

Attribute name	Description	Attribute type
hot	number of "hot" indicators (hot: number of directory accesses, create and execute program)	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1=successful login; 0=unsuccessful	discrete
num_compromised	number of conditions "compromised" (compromised condition: number of file/path not found errors and jumping commands)	continuous
root_shell	1=rootshell active; 0=not active	discrete
su_attempted	1="su root" command executed; 0=not executed	discrete
num_root	number of root accesses	continuous
num_file_creations	number of file creation processes	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of access controlled files modified	continuous
num_outbound_cmds	number of outbound commands within FTP session	continuous
is_hot_login	1=hotlist login; 0=not on hotlist	discrete
is_guest_login	1=guest login; 0=not on guest login	discrete

Features related to time are referred to as "traffic" features. Traffic features in the data set are divided into two groups: same host and same service groups (Lee & Stolfo, 2000).

- **Same host features** are the connections for the previous two seconds that share the same destination host with the current connection
- **Same service features** are only the connections in the previous two seconds that share the same service as the current connection

To solve the issue of identifying patterns for persistent slow probing attacks, the time features

can be recalculated based on a window of 100 connections instead of 2 seconds. These recalculated features are called connection-based traffic features.

The traffic-based features for the two-second period within the KDD'99 10% data set are represented below in Table 6.5.

Table 6.5: Two-second traffic features within KDD'99 10% data set

Attribute name	Description	Attribute type
count	number of connections to the same host as the current connection in the past two seconds	continuous
Note: The following features refer to these same-service connections.		
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous
Note: The following features refer to these same-host connections.		
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous

Tavallaee *et al.* (2009) discovered additional issues with the KDD'99 data set not previously highlighted by other researchers. The data collected for the KDD'99 data set's false alarm characteristics were never analytically examined while creating the data set. The primary tool used for data collection is also prone to dropping packets when under heavy loads, for which no tests were performed. It is clear that there are several issues with the data set, and some improvements were only made years later by Tavallaee *et al.* (2009).

6.2.2 NSL-KDD data set

For training of the classifier, the NSL-KDD data set was chosen, as it vastly improves on the issues found in the old KDD'99 data set (Tavallaee *et al.*, 2009). To address some of the known issues discussed in the previous section, a revised version of the data sets called NSL-KDD was created. The NSL-KDD data set created by Tavallaee *et al.* (2009) has several improvements over the original KDD'99 data set. The improvements can be summarised as follows:

- To eliminate the unbiased results obtained when classifiers are trained using the KDD'99 data set, the NSL data set has no redundant records in the training set
- To improve the reduction rates, the test data set has no duplicate records

- The records for each difficulty level group are aligned with the records in the original KDD data set, in other words, proportional to the percentage thereof
- Training data set consists of 24 attacks for training and the test data set includes an additional 14 attack types

Since the NSL-KDD data set is an extension of the KDD'99 data set, most of the features and characteristics have been covered in the previous section. The focus of this section is to highlight the changes within the NSL-KDD data set and concludes the feature coding process for the data set. Table 6.6 lists the files with descriptions included in the zip file downloaded [2].

Table 6.6: NSL-KDD data set files

File name	Description
KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format
KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
KDDTrain+_20Percent.ARFF	A 20% subset of the KDDTrain+.arff file
KDDTrain+_20Percent.TXT	A 20% subset of the KDDTrain+.txt file
KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format
KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

In order to fully utilise the data set for the experiments, several changes were made. The .arff files used for machine learning only include normal or anomaly labels and no attack labels as with the original KDD'99 data set. The researcher, therefore, replaced the 42nd field with a field called xAttack, which allows splitting the data set per each attack type. The xAttack values are converted back to binary classification after each split. The full Train20% data set also only classifies either normal or anomalous traffic.

This also includes the process of feature coding, wherein the categorical data e.g. protocol_type are converted to numerical values or binary values when applicable. It is important to note that the algorithm still interprets the numerical values as categories. The following pseudo algorithm can be applied to feature code each categorical value:

```
1 Algorithm coding categorical values is
2 input: dataset record DR,
3 categorical field selected CF from database,
```

[2] Available for download at `http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html`

```
4  output: New fields with numerical values NV
5
6  For each DR send CF value
7  //Function receives CF from DR and creates NV field
8  function codeField(CF)
9  Categories = distinct[CF]
10 features = []
11 for cat in categories
12         binary = (CF == cat)
13         features.append(numerical value)
14 end function
```

Listing 6.1: Feature coding pseudocode

The feature coding process was done using VBA programming and Microsoft Excel. We can summarise the categorical fields within the NSL-KDD data set as done below in table 6.7:

Table 6.7: Categorical fields within NSL-KDD data set

Field name	Number of distinct categories
protocol_type	3
service	66
flag	11
attack	22

The process was applied to each categorical field by assigning a numerical value to each distinct category type within the data set. The only exception was made for the attack type, where the xAttack field was created. It is important to note that these fields are not interpreted by the algorithm as numerical values; they are still considered categorical values. The records represented each attack, which was then classified under the attack category.

With the process completed, the data set was left with numerical values and classifications per attack type. The relevant files were then converted into the ARFF format using the WEKA toolkit. To simplify, the table below summarises the xAttack field within the full modified NSL-KDD Train20% data set:

Table 6.8: xAttack field within Train20% data set

Numerical representation	Attack type	Attacks	Representation
1	DOS	teardrop, smurf pod, neptun, land, back	37%
2	U2R	rootkit, perl, loadmodule, buffer overflow	0.1%
3	R2L	ftp write, guess passwd, inmap, multihop, phf, spy, warezclient, warezmaster	1%
4	Probe	ipsweep, nmap, portsweep, satan	9%
5	Normal	normal	53%

The table also includes the attack representation in the NSL-KDD Train20% data set for

comparison with the KDD'99 10% data set. The full NSL-KDD Train data set with 125 973 records has the same attack representation within the 20% training data set, which only consists of 25 192 records. However, the full NSL-KDD data set has 368 048 less records than the 10% KDD'99, a result of removing redundant records.

The NSL-KDD data set has better normal-network representation and a significant decrease in DOS attacks. The modified version of the NSL-KDD Train20% data set was used in the experiments. In comparison, the full NSL-KDD data set clearly has a better attack distribution, and represents real network traffic better than the KDD'99 data set. To provide comparative results, the attack representation was also kept within the sampled data for training the classifier.

The final modified NSL-KDD data sets used in the experiments are available on the Github site: `https://github.com/FransHBotes/NSLKDD-Dataset`. It is important to note that after the data were split between attack types, the xAttack field values were converted back to binary classification, i.e. 0 for normal traffic and 1 for attack traffic.

Once the training data sets were completed, the data sets for validation were created. The pre-processed and feature coded 20% training data set was split into two sections. Section one represents 66% of the original data, and section two represents 34%; the latter is used for validation. Splitting the training data set is a common approach for intrusion detection classifier validation (Noureldien, Hussain & Khalid, 2013; Chand, Mishra, Krishna, Pilli & Govil, 2016). The original attack representation was kept for the training and validation set. The first experiment cycle only used the data sets created for validation. The network traffic distribution based on the xAttack field in the validation data set is summarised below in table 6.9.

Table 6.9: Validation data set distribution

xAttack	66% Set	34% Set
1	6094	3140
2	7	4
3	138	71
4	1511	778
5	8876	4573

It is important to note the difference between the test and training data sets. The process discussed was performed on both training and test data. To test the classifier, we have to use data not previously known to the classifier when trained. When comparing the full test data set with the Test21 version, by attack category, the outcome is as indicated in table 6.10, on the next page. The Test21 data set includes attacks completely unknown to the training data, which increases the realism of the data set. The researcher elected to use the NSL-KDD Test21 data set. The training data set only underwent the coding process and no other modifications were made.

Table 6.10: NSL-KDD test data set comparison

xAttack	Attack type	Full Test	Test21
1	DOS	25%	22%
2	U2R	0.16%	0.31%
3	R2L	10%	19%
4	Probe	5%	9%
5	Normal	43%	18%
6	New	17%	32%

It is evident that the Test21 data set gives a better 'per attack' category representation and therefore would be an improvement when used for testing purposes. The additional attacks not encountered within the training data set were allocated to their relevant attack types. Table 6.11 summarises the allocation made for unknown attacks within the Test21 data set.

Table 6.11: New attack allocation within NSL-KDD Test21 data set

Numerical representation	Attack type	Attacks
1	DOS	apache2, mailbomb, processtable
2	U2R	httptunnel, ps, sqlattack, udpstorm, xterm
3	R2L	named, sendmail, snmpgetattack, snmpguess, worm, xlock, xsnoop, warezmaster
4	Probe	mscan, saint

The new attacks slightly changed the attack representation within the KDD Test21 data set. With xAttack [6] field factored into each attack type, the representation was as follows per attack type:

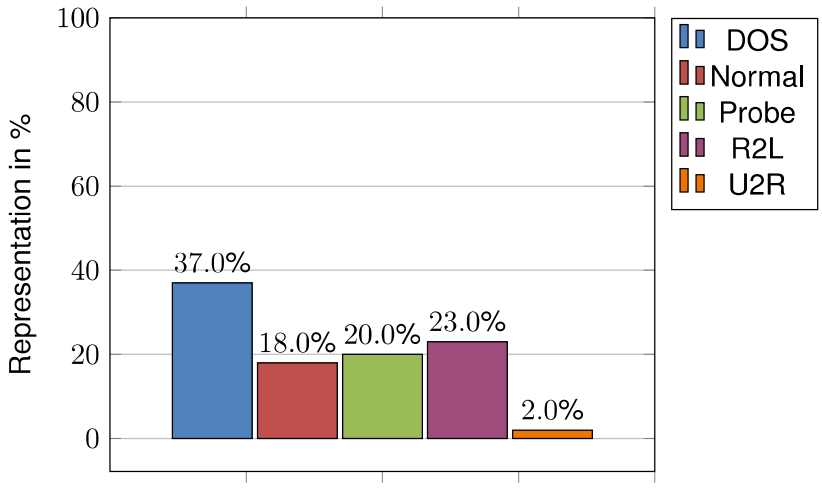

Figure 6.5: New attacks influence on representation within NSL Test21 data set

It is clear that the new attacks increase realism once factored into each relevant attack type. Normal traffic was left unchanged, with a better balance struck between each attack type. Considering the experiment cycles, full 20% training and “per attack” category, the training and test data sets were split as such for each experiment.

To simplify each data set, either training or test was divided to include attack traffic and normal network traffic. With the data sets prepared and coded into numerical values, the next important step was selecting features to use for training. This process was only performed on training data. However, feature reduction was carried out to create the relevant test data set for each feature selected training data set.

6.3 Feature selection

The process of feature selection has been covered in Chapter 3. The purpose of this section is to identify and highlight the features that will perform better detection of each attack type. The feature selection process in this thesis is by no means comprehensive or representing the best possible features. Consideration is made for feature selection as it is important to determine the features relevant to the attack types, as fewer features will result in faster processing.

Some researchers have trained classifiers per attack type (Natesan & Balasubramanie, 2012; Staudemeyer & Omlin, 2014; Petersen, 2015). It can be noted that this eliminates the unfair attack representation, leading to the classifier becoming biased towards certain attack types. The selective experiments allow for a deeper understanding when tasked with detecting specific attacks.

The WEKA tool was used to perform the feature selection process. As discussed in Chapter 3, the information gain and ranker feature selection process was used to extract features relevant to classifying the xAttack field Results from research using similar techniques are discussed for each attack type. The WEKA toolkit offers a simple approach to feature selection. The process followed is outlined below:

1. Open WEKA Explorer.
2. Select the pre-processed .arff file.
3. Open select attributes.
4. For attribute evaluator – InfoGainAttributeEval with default settings.
5. For search method – Ranker with default settings.
6. For attribute selection method – use full training set.
7. The feature for classification is xAttack.
8. The selected features are then selected in WEKA, and the filtered data set is saved.

These steps were followed for each attack type and the full 20% training data sets. The feature selection process was performed on both experiment cycles. The features identified for the full 20% training data set were applied to the validation training data set.

The test data sets were excluded from the feature selection process. To compare the results with other research, the test data must be left in its original form. No features were extracted from the test data set – it can be considered to be excluded. To perform experiments, the feature selected data set and test data set must consist of the same features, therefore feature reduction was performed.

Since none of the data entries were modified, the feature selection test data set and approach can be considered as reliable. Once the process was completed for five training data sets, all with feature selection enabled and pre-processed, the data sets were ready for final experimentation. In the sections below we provide the results gathered from the feature selection process.

A comparison is made between the research carried out by Petersen (2015) on their modified NSL-KDD data set and the research performed by Staudemeyer & Omlin (2014) on the KDD'99 data set. Natesan & Balasubramanie (2012) performed feature selection per attack type on the KDD'99 data set using rough set estimation. The main difference between the feature selection in this thesis and that of Petersen (2015), is that this feature selection process measured each data set split per attack type, whereas the latter used the full combined data set. The modified data set by Petersen (2015) combines the test and train data sets to extract features.

We further discuss the relevant features per attack type in the sections to follow. The exact output from WEKA was added to the thesis for discussion. The selected features were also compared with other relevant research when possible. Table 6.12 outlines all the features within the NSL-KDD training data set on which feature selection was performed.

Table 6.12: All features within NSL-KDD

#	Description	#	Description	#	Description	#	Description
1	Duration	12	su_attempted	23	srv_serror_rate	34	dst_host_srv_diff_host_rate
2	src_bytes	13	num_root	24	rerror_rate	35	dst_host_serror_rate
3	dst_bytes	14	num_file_creations	25	srv_rerror_rate	36	dst_host_srv_serror_rate
4	Land	15	num_shells	26	same_srv_rate	37	dst_host_rerror_rate
5	wrong_fragment	16	num_access_files	27	diff_srv_rate	38	dst_host_srv_rerror_rate
6	Urgent	17	num_outbound_cmds	28	srv_diff_host_rate	39	protocol_type
7	Hot	18	is_host_login	29	dst_host_count	40	service
8	num_failed_logins	19	is_guest_login	30	dst_host_srv_count	41	flag
9	logged_in	20	Count	31	dst_host_same_srv_rate		
10	num_compromised	21	srv_count	32	dst_host_diff_srv_rate		
11	root_shell	22	serror_rate	33	dst_host_same_ src_port_rate		

The screenshot below shows the interface and selected settings:

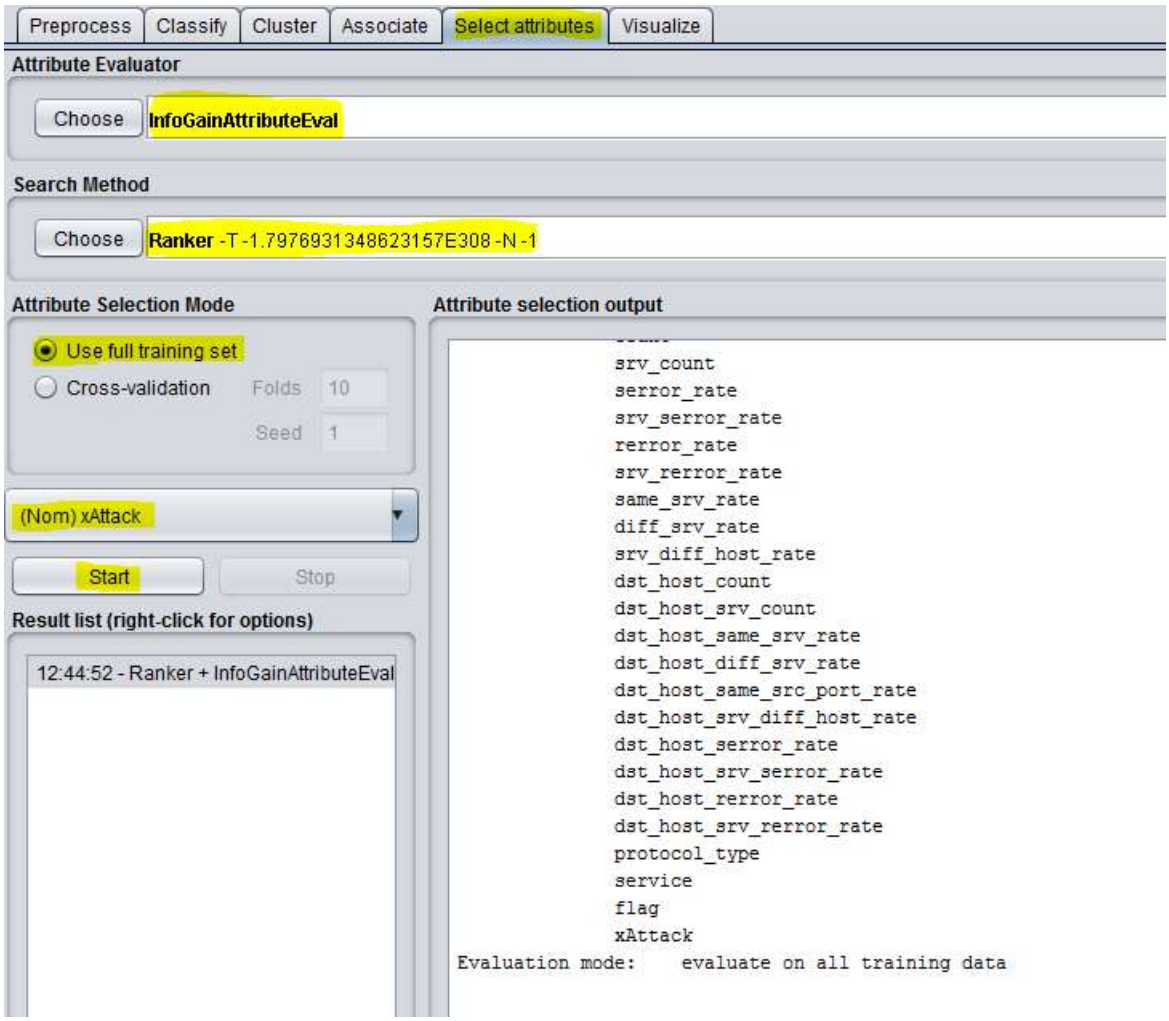

Figure 6.6: WEKA feature selection screenshot

6.3.1 Probe

The following features relevant to Probe attacks were identified through information gain and ranker-based feature extraction:

Table 6.13: Probe features selected

Attack Type	Features (#)
Probe	2, 40, 3, 30, 34, 9, 33, 32, 31, 38, 37, 41, 24, 25, 29, 39, 20, 26, 28, 27, 21, 22, 35, 1, 36

For detailed output on the feature selection process refer to Appendix A and section A.1. Petersen (2015) extracted features relevant to each attack type using the information gain and ranker technique. Petersen (2015) revealed a zero information gain for features 17 and 6 using their custom data set. When compared to the original KDD'99 data set, only 14 features were found relevant (Staudemeyer & Omlin, 2014). For this research, features with 1% or lower significance were excluded from the data set and are marked with * in the output in Appendix A.

6.3.2 User to Root

The following features relevant to U2R attacks were identified by means of information gain and ranker-based feature extraction:

Table 6.14: U2R features selected

Attack type	Features (#)
U2R	11, 40, 7, 30, 10, 29, 14, 1, 33, 21, 6, 13, 9, 41, 39, 19, 4

For detailed output on the feature selection process refer to Appendix A and section A.2. The information gain and ranker method revealed features 40, 7, 1, 11 and 14 as the most significant to Petersen (2015). Petersen (2015) revealed a zero information gain related to U2R attacks for 10 features, significantly less than the 24 identified in this research. When using the KDD'99 data set, only 8 features were deemed relevant based on the information gain (Staudemeyer & Omlin, 2014). Due to the vast number of features with zero information gain, only those were removed from this data set.

6.3.3 Remote to Local

The following features relevant to R2L attacks were identified by information gain and ranker-based feature extraction:

Table 6.15: R2L features selected

Attack type	Features (#)
R2L	2, 40, 3, 30, 7, 33, 34, 21, 20, 29, 19, 28, 32, 39, 1, 36, 31, 41, 8, 9, 35, 27, 11, 4

For detailed output on the feature selection process refer to Appendix A and specifically section A.3. The information gain and ranker method revealed 24 relevant features to classify R2L attacks. Petersen (2015) revealed features 2, 3, 40, 1 and 21 as significant based on the information gain to detect R2L attacks. The results compare well with the KDD'99 data set as several features with zero information gain were also considered insignificant (Staudemeyer & Omlin, 2014). Due to several features with no information gain, only the latter were excluded when selecting features.

6.3.4 Denial of Service

The following features relevant to DoS attacks were identified by information gain and ranker-based feature extraction:

Table 6.16: DOS features selected

Attack type	Features (#)
DOS	2, 27, 26, 41, 40, 3, 20, 36, 35, 31, 32, 22, 23, 30, 9, 34, 29, 33, 21, 28, 38, 39, 1, 37, 24, 25

For detailed output on the feature selection process refer to Appendix A and specifically

section A.4. To detect DoS attacks, the information gain and ranker method revealed 26 significant features. When Petersen (2015) applied the information gain process, the top five ranked features were 2, 27, 40, 3 and 41. Petersen (2015)'s experiment identified the only feature with zero information gain as feature 17. This contradicts the three features found in this experiment. Considering the KDD'99 data set, 11 features can be deemed relevant based on the information gain (Staudemeyer & Omlin, 2014). The features with 1% or lower significance were excluded from the data set and are marked with * in the output in Appendix A.

6.3.5 Full 20% training data set

The following features relevant to detect all attacks were identified by means of information gain and ranker feature extraction:

Table 6.17: Full 20% training features selected

Attack type	Features (#)
20% Training	2, 40, 3, 41, 27, 26, 30, 31, 32, 35, 9, 36, 22, 20, 23, 34, 29, 33, 28, 21, 38, 39, 24, 37, 25, 1

For detailed output on the feature selection process refer to Appendix A and specifically section A.5. The relevant features identified were used to extract features from the training portion of the validation data set as well. The features with 1% or lower significance were excluded from the data set and are marked with * in the output in Appendix A. Wahba, Elsalamouny & Eltaweel (2015) compared different feature selection methods and identified 20 features based on the information gain.

When we further analyse the features selected, on average the overall number of features within the data set are reduced by 42%. Table 6.18 summarises the feature reduction for each set.

Table 6.18: Feature selection comparison

Type	Total features	Total reduced (%)
Probe	25	39%
U2R	17	59%
R2L	24	41%
DOS	26	37%
20% Training	26	37%

Comparing all the features selected reveals that in total six features are never selected for training based on the information gain and ranker method. It can be noted that these features are not relevant to the data set and unnecessarily increase the processing overhead. Table 6.19 summarises the features completely excluded based on the feature selection performed.

Table 6.19: Features excluded

Number #	Description
5	wrong_fragment
12	su_attempted
15	num_shells
16	num_access_files
17	num_outbound_cmds
18	is_host_login

6.4 Classifier setup

With the validation experiment cycle, different parameters were customised for the Ant Tree Miner classifier. The following parameters were tuned for the Ant Tree Miner classifier:

- Colony Size – number of artificial ants within each colony
- Max Iterations – ant colony will iterate until a global-best tree is discovered
- Evaporation Factor – factor to which pheromone will evaporate (decrease) for each entry in the pheromone matrix

As with decision trees, there is a trade-off between performance and accuracy, as shown in the results below. By customising certain aspects of the classifier, the optimal settings were determined for classifying intrusion data, based on the validation experiments. The results obtained from the validation experiment can be found in section 6.6.2.

The results for the final (second) experiment cycle are discussed in Chapter 7, and the preliminary results for the initial (first) experiment cycle are discussed below as this leads to the parameters used to perform the final cycle. Therefore, the first experiment cycle can be considered part of the setup process.

The first experiment cycle only used the full 20% training data set. The data set was split into sections of 66% for training and 34% for validation purposes. Refer to section 6.2 for the discussion. The parameter settings were selected based on the research by Otero *et al.* (2012). Increasing the parameters will not improve performance or accuracy much, as the default values have already been extensively tested by Otero *et al.* (2012). The goal of the first cycle is to find the optimal performance-based parameters. A colony size of 5 and evaporation factor of 0.9 perform slightly better than other combinations (Otero *et al.*, 2012).

Otero *et al.* (2012) mentions that on average, 200 iterations need to be performed before the classifier converges. Several versions of the ATM were experimented with, each with different settings in an attempt to tune the parameters.

Each classifier with different parameters is referred to as a version of the classifier, summarised in table 6.20.

Table 6.20: Ant Tree Miner experiment versions

Parameter	ATM - df	ATM - cs	ATM - ef	ATM - mi	ATM - op
Colony Size	5	2	5	5	2
Max Iterations	200	200	200	50	150
Evaporation Factor	0.9	0.9	0.3	0.9	0.9

- **AntTreeMiner-df** – default values were assigned to the classifier. This can be considered the baseline classifier version to which all other variants were compared
- **AntTreeMiner-cs** – colony size was reduced to only 2
- **AntTreeMiner-ef** – evaporation factor was lowered to 0.3
- **AntTreeMiner-mi** – maximum number of iterations was vastly reduced from 200 to 50
- **AntTreeMiner-op** – optimal parameters and final version, based on results discussed below

For each version, the experiment is run 10 times, with different random seed values assigned to initiate the classifier search. The feature-selected training and test data set for validation was used for this research project. The results in subsection 6.6.2 were analysed using the evaluation techniques found in subsection 6.5. The following can be concluded:

- Reducing the colony size vastly reduces the classifier cost while only slightly impacting accuracy; the parameter would favour the reduction of leaf nodes more. The result contradicts the evaluations performed by Rami & Panchal (2012) who state that an increase in colony size decreases accuracy
- Decreasing the evaporation factor builds more accurate classifiers, but it does not warrant the significant increase in cost, as this increases the amount of pheromones used
- Decreasing the maximum number of iterations leads to a slight decline in accuracy slightly. With an exceptional reduction in cost, the parameter will influence the run time significantly. However, when Rami & Panchal (2012) evaluated the influence of parameters, they noted that the accuracy decreased using the ATM classifier whenever the number of iterations reached 1000

The results have identified a further need for scientific investigation and research; however, this is outside the scope of this study. Further research based on the trade-off between maximum iterations and colony size can add more value to the classifiers' knowledge and on how the virtual organism reacts to different parameters. Based on the results, lowering the maximum iterations clearly has the best performance gain.

Regrettably, this creates issues with regard to the error rate and accuracy. To keep the accuracy and error rate favourable both need to be considered. The results also identified a clear link between tree quality and the FAR rate – when the latter decreased, the tree quality increased. Due to the negative effect of decreasing the evaporation factor, the default value was retained. We can infer that whenever the evaporation factor is lowered, the virtual

organisms' pheromone levels increase and allow for higher quality trees.

Unfortunately, the performance cost far outweighs the benefit. Regarding the zoomed ROC analysis, the OP version has the best balance between the EF and MI performer. When comparing the OP version with the DF, the decrease in cost is significant based on the faster run time and fewer leaf nodes. This comes at a cost as DR and FAR rates worsen, but only slightly. The optimal parameters were selected based on the above statements – a balance between colony size and maximum iterations was chosen. The parameters lowered the number of leaf nodes to 130.40, a 34% reduction in total. The optimal changes allowed for a favourable run time, as the total run time improved by 74%. The balanced approach also allowed the error rate to dip under 1% and accuracy to reach the 99% barrier. The results were considered appropriate based on the arguments above.

6.5 Evaluation techniques

In this section, the methods used to evaluate the classifier are briefly discussed. The application of these techniques can be viewed in the next section and the discussion thereof in the next chapter. By no means do these techniques fully represent the evaluation of intrusion detection systems. For each experiment performed the results are averaged for analysis. The techniques in this section were identified by means of a literature review and further discussed and outlined as used in the thesis.

6.5.1 IDS performance metrics and hypothesis testing

We can draw similarities between hypothesis testing and the traditional confusion matrix used to analyse machine learning problems. It is common for a hypothesis to be derived for each machine learning task as explained in the hypothesis statement of the research (Agarwal & Dhar, 2014). An example of a machine learning prediction model is provided as table C.1 in Appendix C.2. A classic confusion matrix can be derived from the above example as shown in table 6.21. For machine learning tasks, a confusion matrix quantifies the outcome of the trained model once it is evaluated against a test data set (Agarwal & Dhar, 2014). The below figure outlines the four regions of a typical hypothesis test problem in the machine learning domain, as described by Evangelista (2005).

DECISION: / **STATE OF NATURE:**	**Reject H_0 (Predict No Intrusion)**	**Do Not Reject H_0 (Predict Intrusion)**
H_0 is False (No Intrusion)	Good Probability = $1 - \beta$ Frequently called **Power** *(The medical community refers to this as **specificity**.)*	Bad – Type II Error Probability = β *(This will be referred to as a **False Positive** – medical community often plots this on the x axis as 1-specificity.)*
H_0 is True (Intrusion)	Bad – Type I Error Probability = α	Good Probability = $1 - \alpha$ Frequently called **Confidence** *(This will be referred to as **True Positive** when discussing ROC curves; an alternative term is **sensitivity**, typically used by the medical community.)*

Figure 6.7: Hypothesis test by Evangelista (2005)

Wu & Banzhaf (2010) state that the effectiveness of intrusion detection systems should be evaluated by its ability to give correct classifications. Their research concludes that the most known performance metric is the True Positive Rate – Detection Rate with the False Positive Rate – False Alarm Rate (Wu & Banzhaf, 2010). Intrusion detection systems should have a high Detection Rate and low False Alarm Rate according to Wu & Banzhaf (2010). Further investigation into these metrics is required to understand the means of evaluation fully. The confusion matrix below represents the four possible outcomes according to the real nature of an event and the prediction of the IDS. The table is compiled based on the outline provided by Wu & Banzhaf (2010).

Table 6.21: Confusion matrix example

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

From the above confusion matrix, we can identify four possible outcomes to each event that is classified (Wu & Banzhaf, 2010):

- True Positive (*TP*) events are intrusions successfully detected by the IDS
- False Positive (*FP*) events are normal or non-intrusive, wrongly classified as intrusive
- True Negative (*TN*) events are normal or non-intrusive and successfully labelled as such
- False Negative (*FN*) events are intrusions that the IDS missed by classifying them as normal or non-intrusive events

On its own, as a matrix, the confusion matrix does not actually represent a suitable metric to evaluate the IDS. Previous research indicates several important measures, discussed below (Wu & Banzhaf, 2010):

Detection Rate (*DR*) or True Positive Rate (*TPR*) is computed as the ratio between the number of attacks detected correctly and the total number of attacks, also sometimes referred to as recall or sensitivity.

$$DR = TP/(TP + FN) \tag{6.1}$$

False Alarm Rate (*FAR*) or False Positive Rate (*FPR*) is computed as the ratio between the number of normal instances classified as attacks and the total number of normal instances.

$$FAR = FP/(TN + FP) = 1 - TN/(TN + FP) \tag{6.2}$$

Accuracy can be explained as the percentage of instances classified correct. It is important to note that not all of the test data set labels are known to the classifier. Therefore the accuracy rate can be appreciated more than in cases where training and test data are shared, as with this thesis (Tavallaee *et al.*, 2010).

$$TN + TP/TN + TP + FN + FP \tag{6.3}$$

The error rate can be considered as the instances the classifier predicts wrong.

$$(FP + FN)/(TP + TN + FP + FN) \tag{6.4}$$

The F-measure numerates a balance between precision and recall. We can infer that it measures the accuracy of a test. This measure is useful when test data sets are used, especially with unbalanced data sets. Unbalanced data sets tend to have a high accuracy, low error rate but high FAR, by favouring the most common class (Weng & Poon, 2008). Therefore, in these cases the f-measure can be considered a better indicator.

$$F\_measure = \frac{2 \times (TP/(TP + FP)) \times (TP/(TP + FN))}{(TP/(TP + FP)) + (TP/(TP + FN))} \tag{6.5}$$

When evaluating the intrusion detection system, results must be interpreted and not just displayed, as this leads to one-sided evaluations based on only the accuracy of a classifier. Therefore we can infer that an effective and efficient intrusion detection system should maximise the DR and minimise the FAR. Due to the test data set revealing attacks not previously known, we also rely on the accuracy of the classifier and F-measure where required.

6.5.2 ROC curves

The ROC analysis is performed in conjunction with the performance measures, providing illustrative support for finding the parameters that best balance the DR with the FAR. It is important to understand that traditional ROC analysis balances by means of adjusting the operating or decision threshold (Bradley, 1997).

The ATM classifier only produces factor outcomes without any probability score for the decision made. This, however allows the ROC curve to only be plotted for one point $(x, y) = (FP, TP)$. A decision tree, however, has no threshold to tune, and therefore the ROC analysis is purely a graphical representation to identify the best balance.

The method is used quite common with detection and estimation theory van Trees (2002), and is further supported by Axelsson (2000) in his attempt to tie detection and estimation theory to intrusion detection domains. It is crucial to note with ROC analysis that the points $(0:0!)$ and $(1;1!)$ are members of any intrusion detector. It is evident that if all events are intrusions, the DR would be 1.

Unfortunately, this creates another problem in which the algorithm incorrectly classifies all benign packets as intrusive, which in effect consequently leads to the FAR being one as well. Conversely, the same would happen if we pose the rates as 0. Detection and estimation theory values the rule that the DR and FAR rates are linked (van Trees, 2002).

Ostensibly we can adduce that the more events are classified as intrusive, essentially relaxing the criteria on what constitutes an intrusion, the DR will increase but also misclassify more of the malicious packets, hence increasing the FAR. Axelsson (2000) identifies this phenomenon as the base-rate fallacy problem, for the factor limiting the performance of the IDS is not the ability to identify behaviour correctly as intrusive, but rather its ability to suppress the FAR. Shrewdly one would, however, then measure the FAR in relation to how many intrusions are expected to be detected, and not in relation to the maximum number of potential false alarms.

In this research, the ROC employed plots the balance for each classifier under validation. With no threshold to tweak, the assigned parameters however would influence the final result. The graph below visualises how to evaluate the classifiers based on the ROC analysis:

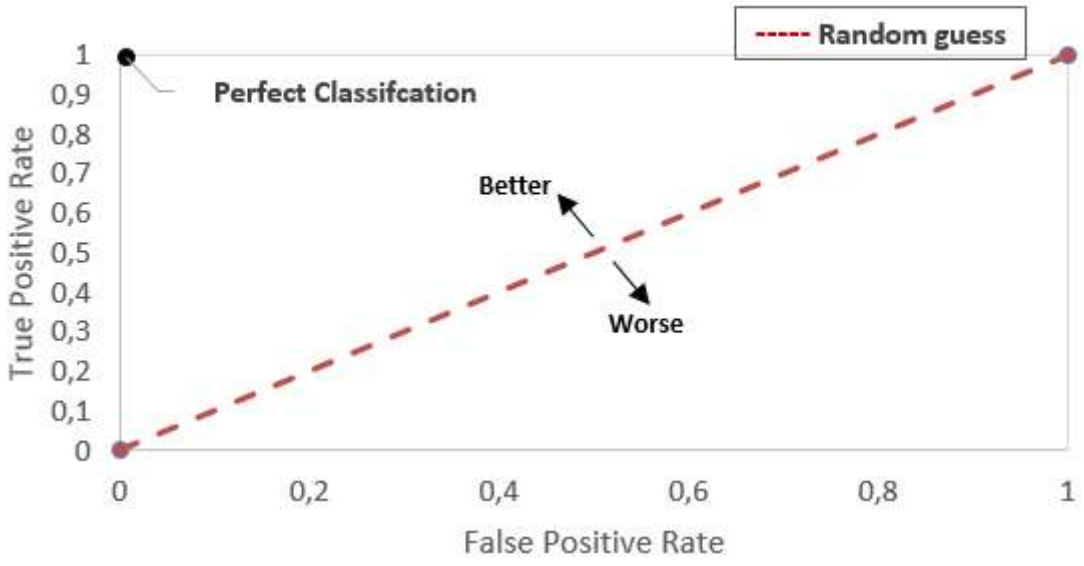

Figure 6.8: ROC analysis example

6.5.3 Cost analysis

Axelsson (2000) iterates the importance of computational complexity within intrusion detection, drawing specific focus to the resources, storage and, specifically, time if the system can perform in real-time. It is important to note that the purpose of this research is to establish

ATM as a classifier for intrusion detection, either real-time or offline. The computational complexity and comparison with other classifiers were performed by Otero *et al.* (2012). The core of cost analysis for this thesis was the results observed for time and the complexity of the decision trees constructed.

Several research projects using machine learning and intrusion detection include the time in seconds as comparison result (Tavallaee *et al.*, 2010). One can argue that although comparable in that research study, the use thereof does not really trade further due to different computer setups, e.g. processing power and algorithm design will result in incomparable results (Tavallaee *et al.*, 2010).

The statements can be disputed, especially in research similar to this research project. The experiment and research do not focus on the design or creation of a commercial-based intrusion detection system, but instead on the ability of the classifier to improve upon previous techniques used. Thus, the computational overhead for comparing results was not such a concern in the research. Some evaluation of the computational cost for the classifier has been done to ensure the best performance.

To evaluate the cost of the classifier, a cost score was designed based on the validation experiments. We argue that the transparent design motivates reproducibility for use as an evaluation technique within future research using the ATM classifier. Ling, Yang, Wang & Zhang (2004) emphasise that to obtain a minimalistic cost with decision trees, fewer leaf nodes are required.

The time taken to construct the decision tree should logically also be considered since less time would naturally require less computational resources. Another important metric is the quality of the decision tree. The quality of a decision tree is based on the relationship between training examples and the error rate. For ATM, the tree quality influences the number of pheromones required to produce the model (Otero *et al.*, 2012). During the validation experimentation it was noted that a higher quality tree uses more pheromones. From the above discussion we can infer that the number of leaf nodes, tree quality and time taken can be considered as metrics for cost evaluation. We can calculate the cost score CS for each experiment by using

$$\begin{aligned} CS &= (\log(LN) \times \frac{T}{60})/(1 - TQ) \ , \\ Given\ &(|1 - TQ| > \epsilon) \parallel (\epsilon = 1 \times 10^{-6}) \ , \end{aligned} \tag{6.6}$$

where LN is the total number of leaf nodes within the model and T represents the time taken to build in seconds, while TQ is the total tree quality for the decision tree. In cases where $TQ = 1$, a highly unlikely event, however theoretically possible, a default value should be applied where $TQ = 99.99999\%$. The default value for TQ as $1 - 1 \times 10^{-6}$ is randomly selected. From the formula we can deduce that the lowest score will represent the best cost metric.

6.6 Results

6.6.1 Introduction

The sections to follow provide the results obtained from both experiment cycles. The results obtained from experiment cycle one and experiment cycle two are summarised in a table and graph in each subsection that follows.

The first experiment cycle is discussed in this chapter, in section 6.4. The second experiment cycle is discussed and analysed in Chapter 7 as it is the cornerstone of the thesis. For a combined overview of the results obtained for the second experiment cycle, refer to Appendix B. Table 6.22 outlines where each statistic gathered from the experiments is obtained.

Table 6.22: Outline of experiment results

Result title	**Reference**
TP	Evaluation technique - Confusion Matrix
FP	Evaluation technique - Confusion Matrix
TN	Evaluation technique - Confusion Matrix
FN	Evaluation technique - Confusion Matrix
Error Rate	ATM result output
Accuracy	ATM result output
DR	Evaluation technique - Detection Rate
FAR	Evaluation technique - False Alarm Rate
Tree Quality	ATM result output
Total Nodes	ATM result output
F-measure	Evaluation technique - F-measure
Leaf Nodes	ATM result output
Runtime (s)	ATM result output
Tree Instances	ATM result output
Cost	Evaluation technique - Cost Analysis

Each results page concludes with a ROC graph as discussed in section 6.5.2. The graph is plotted based on the results represented in the tabled summary for each experiment.

6.6.2 Validation experiment cycle

The following evaluations were done based on the evaluation techniques. The results are based on the average obtained for 10 experiment runs, each per classifier version. The experiments were only performed on the feature selected validation data set.

Table 6.23: Validation experiment results

Evaluation	DF	CS	EF	MI	OP
TP	4562	4555	4560	4516	4544
FP	32	38	29	83	52
TN	3962	3956	3965	3910	3941
FN	12	19	13	57	29
Error Rate	0.5%	0.65%	0.48%	1.63%	0.95%
Accuracy	99.5%	99.35%	99.52%	98.37%	99.05%
DR	99.75%	99.60%	99.72%	98.75%	99.36%
FAR	0.00789	0.00939	0.00726	0.02079	0.01312
Tree Quality	99.24%	99.13%	99.13%	98.11%	98.82%
Total Nodes	237.2	203.2	263.9	137.2	160.00
Leaf Nodes	197.4	165	215.1	115.7	130.40
Runtime (s)	295.08	98.46	489.98	65.05	76.99
Tree Instances	188.8	190.6	131.1	36.8	139
Cost	1477.21	419.6	2200.97	118.33	230.41

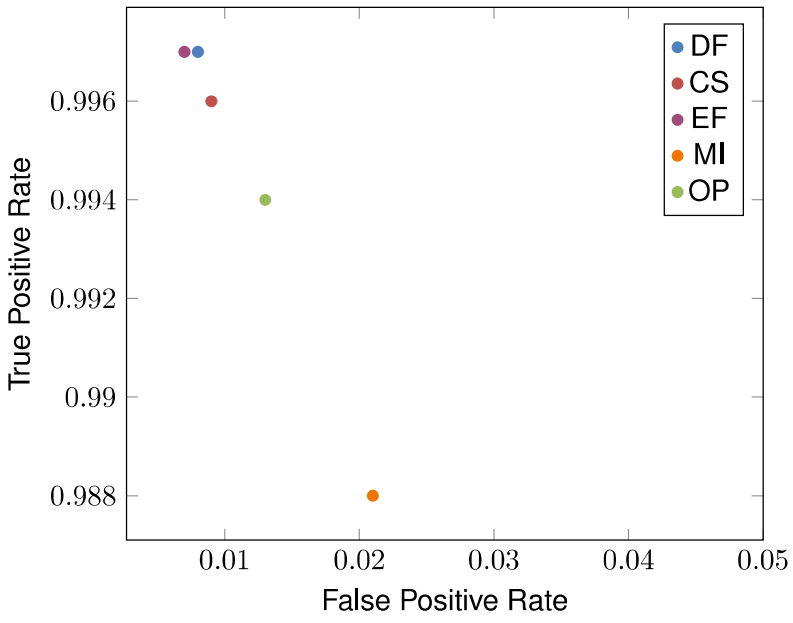

Figure 6.9: ROC analysis – evaluation experiments

6.6.3 Experiment cycle two

The results below are based on the experiments performed for DOS attacks. The feature selected data set is denoted with -FS in the table to follow. The results are based on the average obtained for 10 experiment runs. A discussion of the results follows in Chapter 7.

Table 6.24: DOS experiment results

Evaluation	DOS	DOS-FS
TP	2010	1974
FP	1480	1438
TN	2860	2903
FN	142	179
Error Rate	24.99%	24.89%
Accuracy	75.01%	75.11%
DR	93%	92%
FAR	34%	33%
Tree Quality	99.72%	99.69%
Total Nodes	98.6	111.2
F-measure	0.71	0.70
Leaf Nodes	83.1	96.2
Runtime (s)	157.89	134.27
Tree Instances	138.6	136.7
Cost	1819.41	1419.46

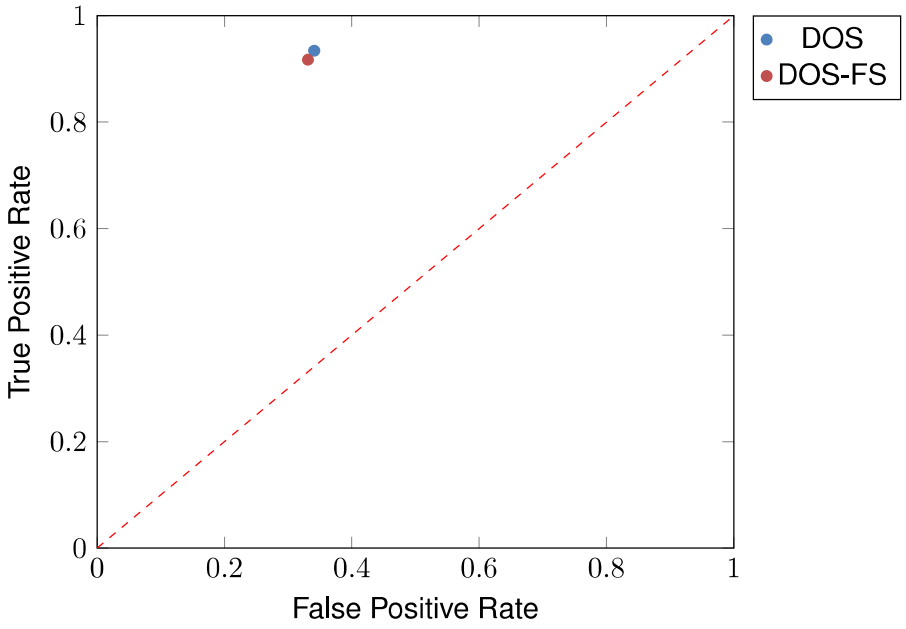

Figure 6.10: ROC analysis – DOS experiment

The results below are based on the experiments performed for probe attacks. The feature selected data set is denoted with -FS in the table to follow. The results are based on the average obtained for 10 experiment runs. A discussion of the results is provided in Chapter 7.

Table 6.25: Probe experiment results

Evaluation	Probe	Probe-FS
TP	1911	1942
FP	760	821
TN	1642	1582
FN	241	210
Error Rate	21.99%	22.63%
Accuracy	78.01%	77.37%
DR	88.79%	90.24%
FAR	32%	34%
Tree Quality	99.39%	99.42%
Total Nodes	84	130.2
F-measure	0.79	0.79
Leaf Nodes	63.5	107.4
Runtime (s)	61.45	59.75
Tree Instances	126.1	133.3
Cost	303.34	347.52

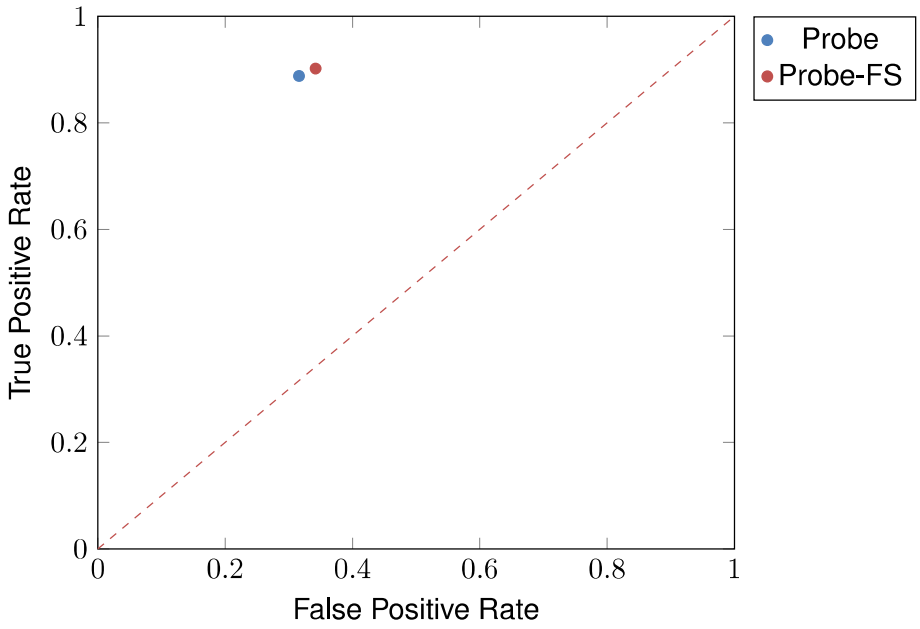

Figure 6.11: ROC analysis – Probe experiment

The results below are based on the experiments performed for Remote to Local (R2L) attacks. The feature selected data set is denoted with -FS in the table to follow. The results are based on the average obtained for 10 experiment runs. A discussion of the results is provided in Chapter 7.

Table 6.26: R2L experiment results

Evaluation	R2L	R2L-FS
TP	2114	2113
FP	2518	2490
TN	236	264
FN	38	39
Error Rate	52.10%	51.56%
Accuracy	47.90%	48.44%
DR	98.22%	98.17%
FAR	91%	90%
Tree Quality	99.55%	99.56%
Total Nodes	49	41.8
F-measure	0.62	0.63
Leaf Nodes	35.2	28.2
Runtime (s)	20.1	20.35
Tree Instances	106.9	122.8
Cost	114.51	110.65

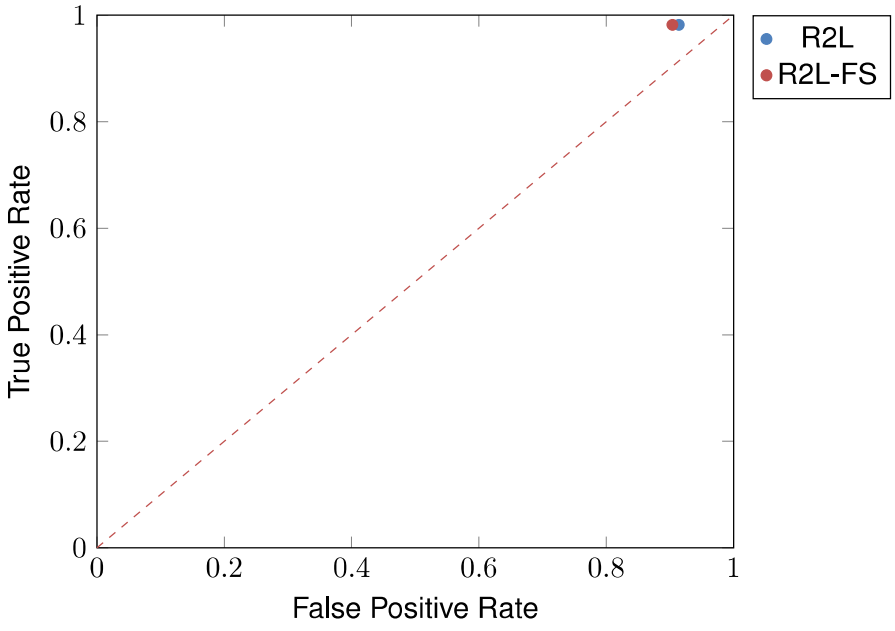

Figure 6.12: ROC analysis – R2L experiment

The results below are based on the experiments performed for User to Remote (U2R) attacks. The feature selected data set is denoted with -FS in the table to follow. The results are based on the average obtained for 10 experiment runs. A discussion of the results is provided in Chapter 7.

Table 6.27: U2R experiment results

Evaluation	U2R	U2R-FS
TP	2149	2150
FP	185	193
TN	17	9
FN	3	3
Error Rate	7.99%	8.31%
Accuracy	92.01%	91.69%
DR	99.86%	99.88%
FAR	92%	96%
Tree Quality	99.91%	99.91%
Total Nodes	5	6
F-measure	0.96	0.96
Leaf Nodes	3	3.5
Runtime (s)	2.04	1.74
Tree Instances	10.5	55.8
Cost	17.69	17.29

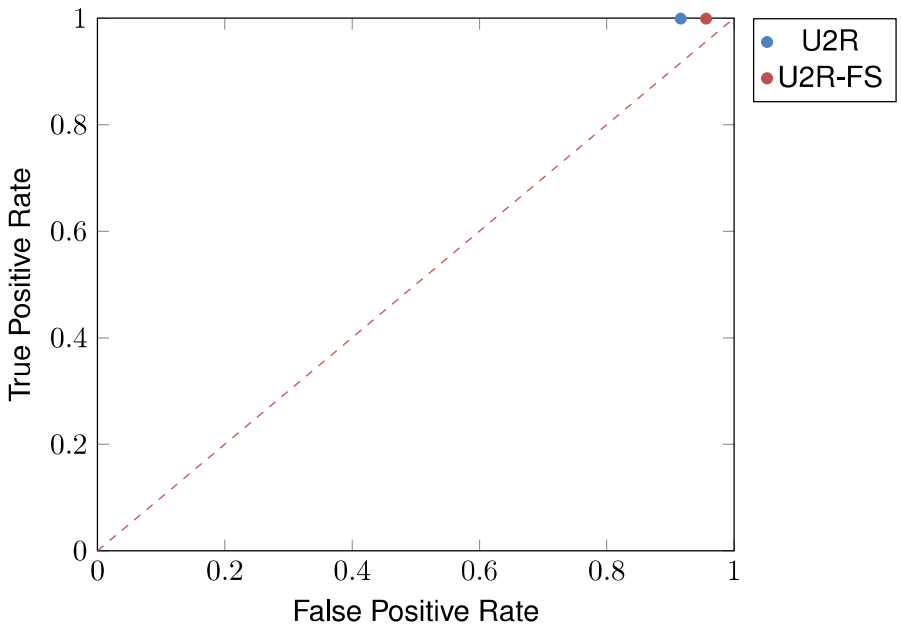

Figure 6.13: ROC analysis – U2R experiment

The results below are based on the experiments performed on the 20% training data set containing all attack types. The feature selected data set is denoted with -FS in the table to follow. The results are based on the average obtained for 10 experiment runs. A discussion of the results is provided in Chapter 7.

Table 6.28: 20Train experiment results

Evaluation	20Train	20Train-FS
TP	1828	1810
FP	4383	4287
TN	5315	5411
FN	324	342
Error Rate	39.72%	39.06%
Accuracy	60.28%	60.94%
DR	84.94%	84.11%
FAR	45%	44%
Tree Quality	98.91%	99.00%
Total Nodes	239.4	196.9
F-measure	0.44	0.44
Leaf Nodes	207.1	158.2
Runtime (s)	241.46	239.09
Tree Instances	143	140.9
Cost	852.67	875.73

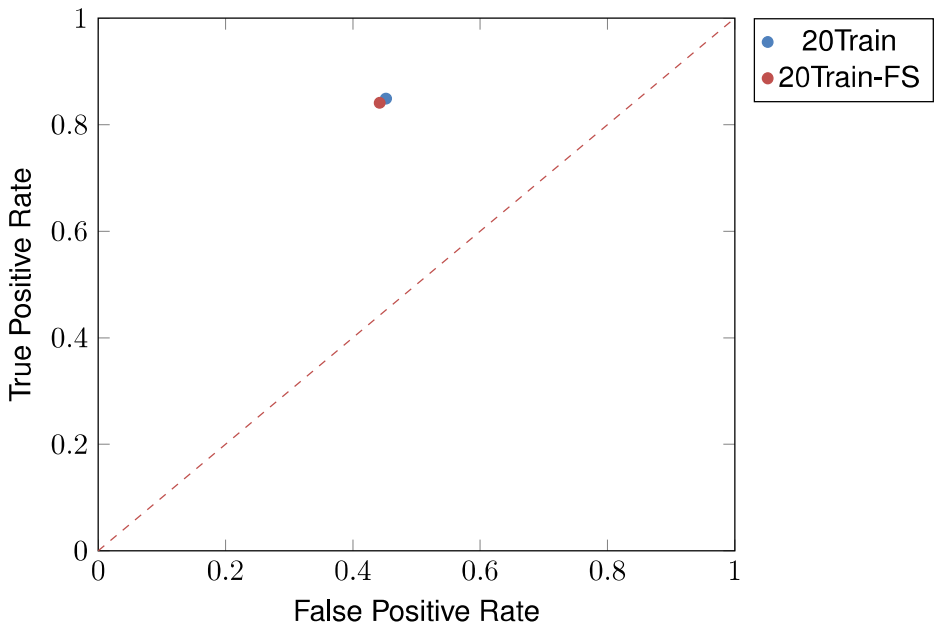

Figure 6.14: ROC analysis – 20Train experiment

6.7 ATMa ensemble

To highlight the potential of ensemble techniques (discussed in Chapter 3), an ensemble version of ATM was created, referred to as ATMa. The technique is similar to the popular bagging predictions approach used in the majority of Kaggle[3] competitions. ATMa builds three prediction models and then creates a final prediction model based on a threshold formula. The final prediction model was tested against the Test21 data set and should be compared to the ATM 20Train results. The experiment, therefore, forms part of experiment cycle two. The threshold formula can be explained using two formulas. We discuss the results in Chapter 7. For calculating A,

$$A = \frac{1}{3}\sum_{i=1}^{3} n_i \ , \tag{6.7}$$

where n is the prediction model, i the index of a single model within the set and n_i the element's value. The final prediction P can be calculated using the average obtained from A as

$$P = f(A) \begin{cases} 1 & > 0.1 \\ 0 & \leq 0.1 \end{cases} , \tag{6.8}$$

where P is the final binary prediction returned (e.g. malicious or normal) for the given record and A is the average of predictions for three models. Due to ensembling the already predicted models, the values that influence the cost metrics are calculated differently. The Tree Quality, Total Nodes, Tree Instances and Leaf Nodes are calculated based on the average of the three models. The run time is the total time in seconds to train all three models. No feature selection was performed to build ATMa, and the prediction models were built based on training with the 20% training data set. The ATMa results are displayed with the 20Train experiment results, which exclude feature selection.

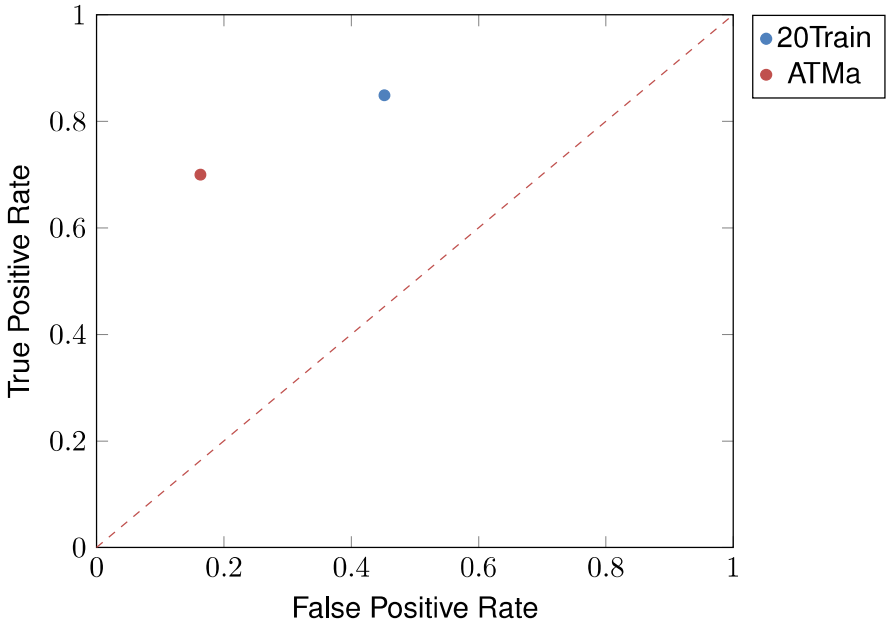

Figure 6.15: ROC analysis – ATMa experiment

[3]https://www.kaggle.com

Table 6.29: ATMa experiment results

Evaluation	20Train	ATMa
TP	1828	6787
FP	4383	350
TN	5315	1803
FN	324	2911
Error Rate	39.72%	27.52%
Accuracy	60.28%	72.48%
DR	84.94%	69.98%
FAR	45%	16%
Tree Quality	98.91%	98.97%
Total Nodes	239.4	308
F-measure	0.44	0.81
Leaf Nodes	207.1	275.67
Runtime (s)	241.46	662.55
Tree Instances	143	145
Cost	852.67	2621.57

6.8 Methodology discussion

Research and experimentation in the combined intrusion detection and machine learning fields have come under much criticism (Tavallaee *et al.*, 2010; Sommer & Paxson, 2010). Poorly designed experiments are to blame for the success discrepancy between the research areas and the practical implementation thereof.

A lengthy chapter has summarised all the necessary choices and steps followed in this research and the experimentation process. To further the quality and highlight the importance of research in the intrusion detection domain, some focus needs to be placed on where previous studies have failed.

Tavallaee *et al.* (2010) find that with their survey research, more than half of the research related to the machine learning and intrusion detection domains does not specify which data sets are used for training and test purposes. The data sets, preparation and use thereof were discussed in rigorous detail.

Many research studies rely on data sets biased towards abnormal activity and high accuracy scores for machine learning research (Tavallaee *et al.*, 2010). The NSL-KDD data set has been used and can be considered the best available data set for research combining the two domains. The improved attack distribution was discussed, and one can argue that the data set vastly improves the research's reliability.

Only 84% of research combining the two domains defines the initial parameters used within the experiment for the classifiers (Tavallaee *et al.*, 2010). The effect is worsened by only 12% motivating the selection of parameters when defined (Tavallaee *et al.*, 2010). Specific focus on parameters, the impact and motivation have been discussed and defined in this research project.

Only 20% of the research states the number of simulations and less than 60% indicates when the results have been averaged based on several evaluation runs (Tavallaee *et al.*, 2010). The process and experiment cycle were detailed. The evaluation techniques and process were clearly defined for each experiment in Chapter 6.

Tavallaee *et al.* (2010) outlines the lack of evaluating classifiers based on each attack type, as only 26% of research has done so. This research project presented and evaluated each attack type present in the data set separately.

Performance analysis regarding the memory, time or cost overheads is important to critical for IDS (Tavallaee *et al.*, 2010). Only 19% of the papers surveyed studied the performance of the methods introduced and only 40% indicated the hardware used (Tavallaee *et al.*, 2010). In this research, specific focus was placed on the cost evaluation based on the ATM classifier's performance. The evaluation technique and hardware used were outlined in detail, allowing for comparability and reproducibility.

Much emphasis has been placed on the validity and reliability of the research experiments. Based on the discussion and the detailed outline for every decision and process, this research can be considered valid and reliable. The detailed process allows for high reproducibility of the entire research project. The research performed is valid, reliable and reproducible, therefore, the experimentation process is considered to be of a high quality and successful. With the experimentation completed, the focus shifted to the analysis and discussion of the results in section 6.6.

CHAPTER 7

RESULT ANALYSIS AND SUMMARY

7.1 Introduction

In this section, we discuss the experimental results and compare it with other studies. The results for the second experiment cycle are scrutinised and compared, where possible, to other research. The significance of conducting reliable, reproducible and transparent research has been highlighted briefly in section 6.8.

Before we discuss the results, further insight into the research domain is required. Several studies on classifiers for intrusion detection promise detection and accuracy rates of over 99%, but this needs to be considered in the context of how the experiments were performed. Without applying the basic principles, these studies have become almost impossible to replicate or compare.

We argue that there is a definite issue within the cross-domain research areas to provide comparable results, and this statement is supported by Tavallaee *et al.* (2009) and Sommer & Paxson (2010). Therefore, finding exact comparable results is hard, if not impossible, in some instances. While creating the NSL-KDD data set, Tavallaee *et al.* (2009) also experimented with several classifiers on their Test21 data set.

For a detailed analysis and motivation on the data set used in this research, refer to Chapter 6. We can compare the Test21 results from Tavallaee *et al.* (2009) with the results obtained in the previous chapter (refer to table 6.19 and table 6.29). The experiments referred to as the 20Train and ATMa used the same training and test data sets as Tavallaee *et al.* (2009).

The first experimental session referred to as the validation experiment cycle was discussed in section 6.4. The validation experiment cycle only referred to the validation of the algorithm and is therefore excluded from further analysis in the discussion to follow. To ease the discussion of the results obtained, the chapter is split into two sections, namely general discussion and discussion of training per attack type.

7.2 General discussion

The purpose of the 20Train experiment was to identify the ATM classifier's ability to classify all attack classes as either normal or malicious network traffic. The ATM's ability to classify the full Test21 data set by training with only 20% of the NSL-KDD training data can be considered adequate with an accuracy rate of 60% and 61% respectively. The high accuracy rates noted with experiment cycle one are significantly reduced, as the Test21 data set includes hard to detect attacks and unseen attacks (Tavallaee *et al.*, 2009). This was explained in more detail

in Chapter 6.

The difficulty of the test data set explains the significant drop in accuracy across the board e.g. 99% vs. 60%. During experiment cycle one when we averaged the results, a 99% accuracy (refer section 6.6.2) was observed. In the validation section, the data sets were split in order to create the validation data set. Cases where cross-validation or the split of a training data set (as with experiment cycle one) takes place, are not comparable with one another due to the selection process randomly selecting the records when the validation data sets are created. Therefore, the differences in the creation of validation data sets makes the results obtained incomparable, especially with actual test data sets where the records were not randomly selected. Experiment cycle one validates the ATM's performance on more known network traffic as the actual training data set is split (refer Chapter 6). The results obtained from the Test21 data set clearly outline the above mentioned phenomenon as the ATM obtained an accuracy rate of 60% and 61% respectively due to the difficult test data set. The difficulty is increased, with reference to table 6.10, as 32% of the network traffic within Test21 is completely new to the classifier and could be considered similar to zero-day attacks.

The high detection rate of 84% notes the ATM classifier's ability to detect attacks correctly. From an anomaly intrusion detection perspective the detection rate is unconvincing, as 16% of all attacks would not be detected by the classifier. Another detection component would be required to detect the residual attacks.

The false alarm rate introduces further concerns as the ATM noted high false alarm rates with 44%. Based on analysing the ROC graph, figure 6.14, the classifier scores well above the random guess phenomenon. Although the ATM reflected a high ability to identify attacks and the lack of ability to suppress the false alarm rate, based on only the ROC analysis we can consider the classifier to be adequate.

By using feature selection, the ATM's evaluation metrics improved, but with a negative effect on the cost due to the improved tree quality. Another example is the classical machine learning trade off between accuracy and computational cost. We argue that the increased cost is trivial as the classifier noted improved run time and used fewer leaf nodes with feature selection implemented. The cost overhead relates indirectly to the pheromones and processing power required to produce the improved tree quality.

When we consider the influence of feature selection on all experiments the following can be noted: Feature selection reduced the run time in 80% of the experiments, while the overall cost metric improved for only 60% of the experiments performed. The improved performance is a direct result of having fewer features to process. The effect of feature selection on the accuracy is still debatable as the accuracy decreased in 40% of the experiments.

It can be noted that the overall improvement in accuracy is due to fewer features and the retention of quality features for classification. The statement is supported by the results as the tree quality improved in 80% of the experiments. Based on the results obtained we can note

that the feature selection has a positive influence on the ATM's ability as an intrusion detection candidate.

To improve the results obtained, an ensemble version of the ATM classifier was built (refer to section 6.7 for more information). The ATMa classifier experiment compares directly with the 20Train experiment performed as both use the same training and test data sets. ATMa significantly improves on the ATM 20Train results in terms of the accuracy, error rate and the false alarm rate. The improved results come at a cost; the run time significantly increased as the model was built three times within the ensemble. The increased F-measure of 37% highlights an improved balance in the precision of the classifier.

It was noted in Chapter 6 that a good intrusion detection system should maximise the detection rate and minimise the false alarm rate. Although the detection rate decreases with ATMa by 15%, the false alarm rate is minimised by 29%. The false alarm rate of only 16% highlights potential as an intrusion detection classifier with further improvement. It is important to note that the Test21 data set is considered an anomaly detection data set (Tavallaee *et al.*, 2009). ATMa improves the accuracy and error rates due to the underlying threshold formula discussed in Chapter 6.

The threshold formula favours malicious predictions and therefore decreases the false alarm rate. To simplify, let us consider a prediction model from three sets as S, $S = \{1, 0, 0\}$, is identified. The ensemble prediction would result in 1, classified as malicious. In effect, ATMa will only classify traffic as normal if all three predictive models agree. The threshold value is currently set randomly at 0.1 and is another avenue for further research. When we compare ATMa with all the experiments in cycle two (refer to Appendix B), the classifier scores the best false alarm rate and worst detection rate – an example of the base-rate fallacy problem coined by Axelsson (2000).

Based on the computational cost, the ATM and ATMa classifiers would not be adequate for real-time network analysis and better suited for an offline intrusion detection system. It is essential to compare the results obtained with other research to further examine the experiment results. The result highlights a clear route for improvement – with ensemble techniques we argue that further improvement to the cost of the classifier is required. The overall positive influence of feature selection can improve the computational cost of the ATMa classifier.

Table 7.1 outlines the results of Tavallaee *et al.* (2009), evaluating several machine learning classifiers within their Test21 data set. For a combined summary of all results obtained from experiment cycle two, refer to Appendix B. A limited amount of statistics is available from other research performed. We therefore solely rely on accuracy as a metric for comparison. Ensemble classifiers are denoted with * in table 7.1.

Table 7.1: Accuracy of several classifiers on Test21 data set

Classifier	Accuracy
ATMa*	72.48%
NB-Tree*	66.16%
Decision Tree J48	63.97%
Random Forest*	63.26%
ATM	60.94%
Random Tree	58.51%
Naïve Bayesian	55.77%
M-Layer Perceptron	57.34%
SVM	42.29%

The Decision Tree J48 algorithm employs the C4.5 split. Intuitively, Random Forests, NB-Tree and ATMa can be regarded as ensemble classifiers, i.e. a combination of classifiers or prediction models. Refer to Chapter 3 for more information on ensemble techniques and Chapter 6 for the approach followed with ATMa. Although the ATM combines ant colony optimisation and decision trees, it cannot be considered an ensemble classifier. The ATM induces decision trees based on ant colony optimisation. The ATM can be compared with novel algorithms such as Decision Tree J48, SVM, M-Layer Perceptron, etc. Compared with other novel algorithms the ATM performs favourably with only a 3% lower accuracy than the traditional decision tree classifier. The results obtained by the ATM classifier make a strong case as an intrusion classifier by outperforming several other classifiers. The results support the notion that decision tree-based classifiers are among the best for intrusion detection. ATMa outperforms all the other classifiers with an accuracy of 72%. As for intrusion detection, ATMa is a clear winner among several classifiers due to improved accuracy, reduced false alarms and error rates. We continue the discussion in the next section as we look at the ATM's ability to classify only specific attacks.

7.3 Discussion of training per attack type

In order to have a full overview of the ATM's classification ability on the NSL-KDD data set, the classifier was trained per attack type. This introduced the ATM classifier to highly unbalanced data sets.

Analysing the results obtained shows instances of a high false alarm rate and very low error rates, a classical indication of training on an unbalanced data set. This is due to the algorithm favouring the majority class in the data set. We therefore also rely on the F-measure as discussed in the previous chapter.

Regrettably, comparative results split per attack type using the same training and test data set is scarce. The ATM obtained the best accuracy with classifying U2R attacks (92.01% accuracy); with the highly unbalanced data set the F-measure of 0.96 can be considered a good result. However, for intrusion detection, a balance between the detection rate and false alarm rate is required. We refer to the ROC analysis later in the section.

Revathi & Malathi (2014) experimented with detecting U2R attacks using the NSL-KDD data set, and their multi-layer perceptron obtained the highest result with 88.46% accuracy. Revathi & Malathi (2014) however fail to mention which of the training and test data sets were used. The ATM's ability to detect DOS and probe attacks could be considered good, as this includes the lowest percentage of false alarm rates with high detection rates of over 90%. Within the full Train20% data set, Probe and DOS attacks were among the highest represented and we can note that an adequate amount of training data was supplied. Figure 6.5 highlights the distribution of classes within the Test21 data set. For training a classifier per attack type, the attack would require a sufficient amount of data. R2L and U2R attacks represented in some instances lower than 1% of the supplied training data; however, on average it represents 20% of the distribution within the Test21 data set. It can be argued that with more training data representing all classes or training different models from sections of the training data, the results of the 20Train experiment can be improved. The accuracy of 77.37% for probe attacks can also be considered adequate with the difficult Test21 data set. The overall results for probe attacks are considered good with the potential to improve the false alarm rate. Noureldien & Yousif (2016) evaluated the accuracy of various machine learning techniques to detect DOS attacks using the Train20% and Test21 NSL-KDD data sets.

The ATM outperforms the J48 Decision Tree classifier used by Noureldien & Yousif (2016) by 3%. A spike in the cost is also observed with training DOS attacks due to the very high tree quality and run time recorded. To compare the ROC analysis for each attack type, a graph combining the ROC for all experiments is shown below. Due to the overall positive benefit of feature selection, the analysis is limited to only feature selected experiments.

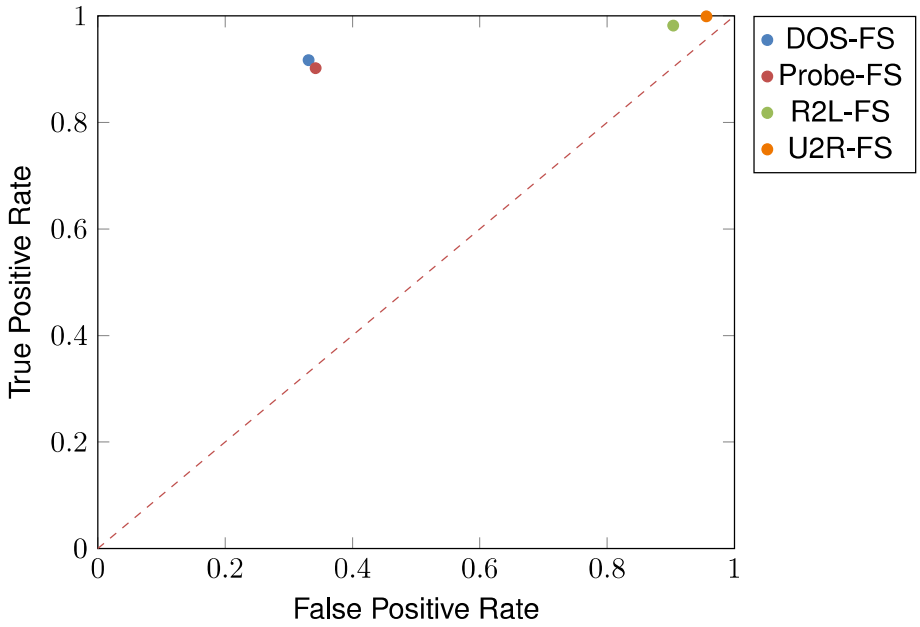

Figure 7.1: ROC analysis – attack type experiments

The ROC analysis for all attack types reveals the inadequacy when detecting R2L and U2R attacks. While the results obtained for accuracy do seem favourable, for intrusion detection the ATM's predictions for R2L and U2R is near random guessing. The ROC results obtained for

DOS and probe attacks are very close to the 20Train and ATMa results. Overall, when we exclude ATMa, the ATM performs better when split per attack type than tasked with detecting all attack types. A high F-measure of over 50% is observed in all cases trained per attack type, indicating that a much better balance is obtained when training per attack type. A closer look at the costs observed also notes very fast training times, and low cost metrics with only DOS attacks the exclusion. Without more training examples the results obtained for R2L and U2R attacks are poor and cannot be considered for intrusion detection.

Based on the results obtained from the experiments, the ATM contested and in some instances improved on traditional decision trees, with the ATMa outperforming both novel and ensemble techniques. We conclude the discussion and thesis in the next chapter.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this chapter, we conclude the work presented, outline the main contributions made and highlight the limitations of the work presented. Additionally, possible future research areas are explored.

8.1 Conclusion

The focus of this research was to implement the ATM within the intrusion detection field. During the research, the ATM has been successfully applied within the intrusion detection field while offering competitive results with other established classifiers. The results obtained are satisfactory and among the top percentile when compared to several other classification techniques.

The ATM scored only 3% lower than the traditional J48 Decision Tree. The results supported notions mentioned by Chennupati (2014) that the true potential of the ATM is within ensemble techniques, as we created the ATMa, an ensemble classifier. The results obtained for ATMa outperformed all other classifiers experimented with by Tavallaee *et al.* (2009) on their NSL-KDD data set. ATMa also outperformed other ensemble decision tree-based classifiers and supports the belief that decision tree-based classifiers are still among the best to be used with intrusion detection. An experiment cycle training the ATM per attack type highlighted the need for more training data for R2L and U2R data sets. The ATM performed well when tasked with detecting DOS and probe attacks, although the high false alarm rates were concerning.

The use of a new cost metric to analyse the results of each experiment allows a baseline for further research, especially improved versions of the ATM classifier. The cost metric introduced in section 6.4 is specifically tailored to the ATM classifier and has shown usefulness when performing experiments.

The validation experiments showed contradicting results to the observations made by Rami & Panchal (2012), as it was noted that the colony size vastly reduces the classifier cost while only slightly decreases the accuracy.

The ATM performed too slow for real-time intrusion detection; instead, it makes a strong argument to be considered for offline systems. The ATMa showed improvements in all areas over the ATM except for cost and should rather be considered for intrusion detection. The ATMa's overall accuracy of 73% with a false alarm rate of 16% far improves on previous results; however, it still cannot be advised for a commercial grade intrusion detection system. Overall, the research decreases the gap between machine learning and the intrusion detection

domain. The transparency and reproducibility within the experiments instantly vanquish the current academic issues found in similar research. We urge researchers to consider applying the same principles and to make use of the cost metrics implemented for analysis.

8.2 Research contributions

During the course of the research, each research question was answered and the hypothesis successfully supported by an overall positive result.

- The baseline provided by decision trees and ant colony optimisation within intrusion detection were highlighted in Chapter 4 and further supported by comparable research in the results discussion in Chapter 6
- Several metrics were chosen to evaluate the classifiers. However, the emphasis was placed on how intrusion detection systems should be evaluated, and specific metrics were chosen in Chapter 6 based on a literature review performed
- The ATM was successfully implemented within the intrusion detection domain and compared with other results in Chapter 6. To further the research contribution, the classifier was trained per attack type to grasp its ability to only classify specific attacks
- The research supports notions by other researchers, as we created an ensemble version ATMa that outperformed the ATM and other classifiers experimented on using the NSL-KDD data set for intrusion detection
- The research introduced a new cost metric to analyse the results obtained by the ATM or equivalent classifiers. The cost metric was designed based on the validation experiments performed and well customised for the ATM. The metric has shown usefulness in evaluating the computational cost of algorithms during experimentation

8.3 Future work and challenges

As this research has shown, ATM and ATMa classifiers offer a number of interesting problems that need to be addressed before they can be used in a commercial system. Some crucial improvements are required to the classifiers and experimentation process.

- **Cost** – the cost of both classifiers need to be improved significantly, based on the researcher's own cost metric. A closer look at minimising the features within the data set can be considered. The feature selection process can be combined with the parameter investigation to maximise the performance gained
- **Ensembling –** the ATMa improves significantly on the ATM. More complex and optimised techniques can be used for ensembling the classifier. The threshold formula can also be further investigated to identify a maximum balance between the detection rate and false alarm rate
- **Data sets –** the initial premise of the research was to establish the ATM within intrusion detection and provide a foundation for further research. The data sets, although considered old, used for training and test phases were chosen due to the comparability with other research. Due to the established foundation, future research should use more realistic data sets or real network traffic. During the time of experimentation, 2016, a

recent data set UNSW-NB15 was made available. During this period limited comparable research was available; future research should include the modern UNSW-NB15 data set

The last avenue for future research is building an intrusion detection system based on the ATM or variant classifiers. It is important to iterate that all other mentioned research avenues must first be explored before considering implementing the classifier for an actual intrusion detection system. To further motivate the ATM classifier for intrusion detection, the operating model in figure 8.1 is proposed.

The model operates in two phases as an intrusion detection system: the training phase and the operating phase. The training phase is first initiated to train a classification model using the ATM. During this phase, ensemble techniques can also be applied to improve the classification accuracy of the initial model.

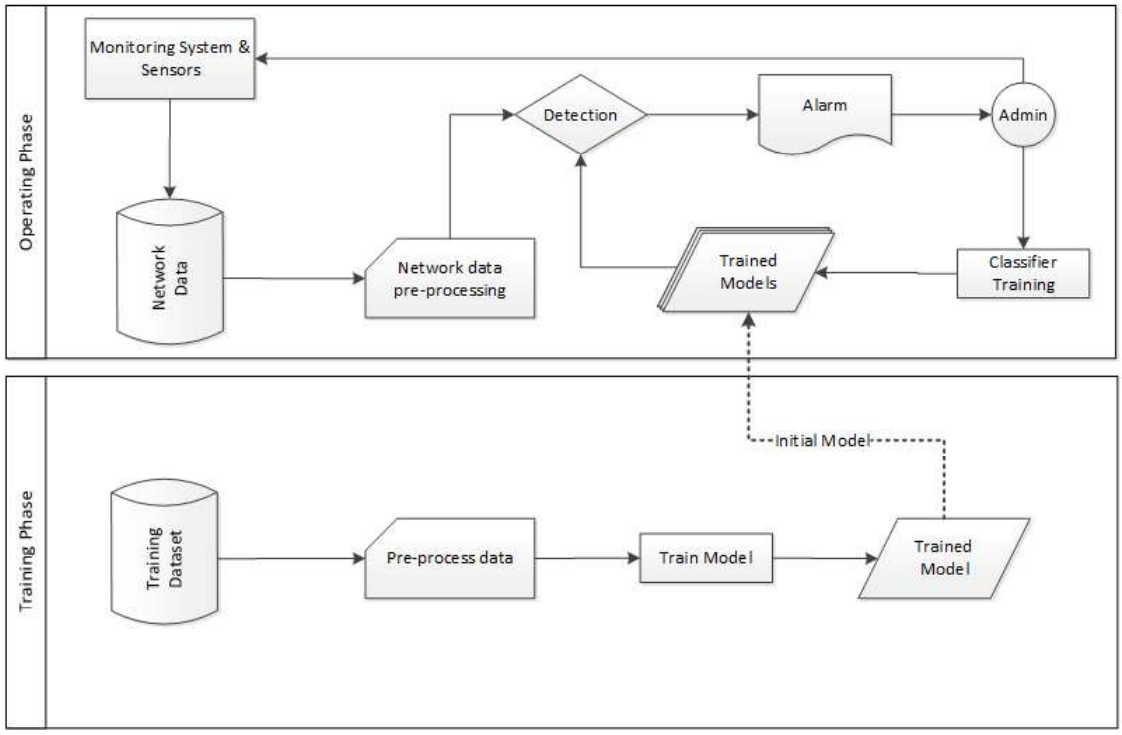

Figure 8.1: Proposed model for intrusion detection

The initial trained model is then applied in the operating phase to be used for detection. Data collected by means of intrusion detection monitors and sensors are fed into a network data database as relevant features are extracted. The classifier is then tasked with classifying the network data based on the current set of trained models. Whenever malicious network traffic is identified, the alarm is sent to an intrusion detection administrator for verification. The administrator then either discards the alarm or feeds the data back into the training phase. As the process continues, trained models will be stacked together by means of an ensemble, and the system will become more accurate as time passes.

Overall, the research decreases the gap between machine learning and the intrusion detection domain. The research supports the notion that machine learning can fill a large void created

by outdated detection techniques and evolving attacks within the intrusion detection field (Sommer & Paxson, 2010; Tavallaee *et al.*, 2010; Chahal & Kaur, 2016; Rudd *et al.*, 2017).

The foundation for implementing ATM in intrusion detection has been established during this research. Based on the results obtained, the researchers are satisfied with the research and excited about future prospects utilising the ATM within intrusion detection.

REFERENCE LIST

Abdelrahman, A.S., Saeed, R.A. & Alsaqour, R.A. 2016. QoS performance study of real-time transport protocol over VoIP. *Journal of Engineering and Applied Sciences*, 11(9):5608–5615. ISSN 1819-6608.

Abdlhamed, M., Kifayat, K., Shi, Q. & Hurst, W. 2017. Intrusion prediction systems. In I.M. Alsmadi, G. Karabatis & A. Aleroud (eds.), *Information fusion for cyber-security analytics*: 155–174. Boston, United States: Springer International.

Abuadlla, Y., Kvascev, G., Gajin, S. & Jovanovic, Z. 2014. Flow-based anomaly intrusion detection system using two neural network stages. *Computer Science and Information Systems*, 11(2):601–622. doi:10.2298/CSIS130415035A.

Agarwal, R. & Dhar, V. 2014. Big data, data science, and analytics: the opportunity and challenge for is research. *Information Systems Research*, 25(3):443–448. doi:10.1287/isre.2014.0546.

Agathou, A. & Tzouramanis, T. 2008. The role of data mining in intrusion detection technology. In G.D. Garson & M. Khosrow-Pour (eds.), *Handbook of research on public information technology*: 463–473. Hersey, United Stated: IGI Global.

Aggarwal, P. & Sharma, S.K. 2015. An empirical comparison of classifiers to analyze intrusion detection. *Proceedings of the International Conference on Advanced Computing and Communication Technologies, Haryana, 21 February 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 446–450. doi:10.1109/ACCT.2015.59.

Aghdam, M.H. & Kabiri, P. 2016. Feature selection for intrusion detection system using ant colony optimization. *International Journal of Network Security*, 18(3):420–432. doi:10.1007/978-3-319-32213-1 27.

Ahmad, I., Abdullah, A.B. & Alghamdi, A.S. 2010. Applying neural network to U2R attacks.
Proceedings of the International Symposium on Industrial Electronics and Applications, Penang, 3 October 2010. New Jersey, USA: Institute of Electrical and Electronics Engineers: 295–299. doi:10.1109/ISIEA.2010.5679451. ISBN 9781424476473.

Ahmad, I., Abdullah, A.B., Alghamdi, A.S., Demiralp, M., Baykara, N.A. & Mastorakis, N.E. 2009. Artificial neural network approaches to intrusion detection: a review. In M. Demiralp, N.A. Baykara & N.E. Mastorakis (eds.), *Proceedings of the International World Scientific and Engineering Academy and Society Conference on Telecommunications and Informatics, Istanbul, 30 May 2009*. New York, USA: Association for Computing Machinery: 200–205.

Akamai. 2016. *State of security report Q3*.
https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/ q3-2016-state-of-the-internet-security-report.pdf [24/01/2017].

Al-Jarrah, O. & Arafat, A. 2015. Network intrusion detection system using neural network classification of attack behavior. *Proceedings of the International Conference on Information and Communication Systems, Amman, 7 April 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6. doi:10.1109/IACS.2014.6841978.

Albayati, M. & Issac, B. 2015. Analysis of intelligent classifiers and enhancing the detection accuracy for intrusion detection system. *International Journal of Computational Intelligence Systems*, 8(5):841–853. doi:10.1080/18756891.2015.1084705.

Alfaro, E., Gamez, M. & Garcia, N. 2013. Adabag: an R package for classification with boosting and bagging. *Journal of Statistical Software*, 54(2):1–35. doi:10.18637/jss.v054.i02.

Alomari, E., Manickam, S., Gupta, B., Karuppayah, S. & Alfaris, R. 2012. Botnet-based distributed denial of service (DDoS) attacks on web servers: classification and art. *International Journal of Computer Applications*, 49(7):24–32. doi:10.5120/7640-0724.

Alonso, S., Cordon, O., De Viana, I.F. & Herrera, F. 2004. Integrating evolutionary computation components in ant colony optimization. In L.N. de Castro & F.J.V. Zuben (eds.), *Recent developments in biologically inspired computing*: 148–180. Hersey, United Stated: IGI Global.

Alrajeh, N.A., Lloret, J. & Loo, J. 2013. Secure routing protocol using cross-layer design and energy harvesting in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 9(1):1–11. doi:10.1155/2013/374796.

Amudhavel, J., Brindha, V., Anantharaj, B., Karthikeyan, P., Bhuvaneswari, B., Vasanthi, M., Nivetha, D. & Vinodha, D. 2016. A survey on intrusion detection system: state of the art review. *Indian Journal of Science and Technology*, 9(11):1–9. doi:10.17485/ijst/2016/v9i11/89264.

Anderson, J.P. 1972. *Computer security technology planning study*. http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf [24/01/2017].

Anwar, S., Mohamad Zain, J., Zolkipli, M.F., Inayat, Z., Khan, S., Anthony, B. & Chang, V. 2017. From intrusion detection to an intrusion response system: fundamentals, requirements, and future directions. *Algorithms*, 10(2):1–24. doi:10.3390/a10020039.

Aqil, A., Atya, A.O.F., Jaeger, T., Krishnamurthy, S.V., Levitt, K., McDaniel, P.D., Rowe, J. & Swami, A. 2015. Detection of stealthy TCP-based DoS attacks. *Proceedings of the Institute of Electrical and Electronics Engineers Military Communications Conference, Tampa, 26 October 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 348–353. doi:10.1109/MILCOM.2015.7357467.

Axelsson, S. 2000. The base-rate fallacy and the difficulty of intrusion detection. *Association for Computing Machinery Transactions on Information and System Security*, 3(3):186–205. doi:10.1145/357830.357849.

Aydın, M.A., Zaim, A.H. & Ceylan, K.G. 2008. A hybrid intrusion detection system design for computer network security. *Computers and Electrical Engineering*, 35(3):517–526. doi:10.1016/j.compeleceng.2008.12.005.

Baiardi, F., Coro, F., Tonelli, F. & Sgandurra, D. 2014. A scenario method to automatically` assess ICT risk. *Proceedings of the Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Torino, 12 February 2014*, 22. New Jersey,
USA: Institute of Electrical and Electronics Engineers: 544–551. doi:10.1109/PDP.2014.105.

Balon-Perin, A. 2012. Ensemble-based methods for intrusion detection. Master's thesis, Norwegian University of Science and Technology, Norwegia.
Banfield, R.E., Hall, L.O., Bowyer, K.W. & Kegelmeyer, W.P. 2007. A comparison of decision tree ensemble creation techniques. *Transactions on Pattern Analysis and Machine Intelligence*, 29(1):173–180. doi:10.1109/TPAMI.2007.2.

Barros, R.C., De Carvalho, A.C.P.L.F. & Freitas, A.A. 2015. *Automatic design of decision-tree induction algorithms*. New York, United States: Springer International.

Bezdek, J. 1993. Intelligence: computational versus artificial. *Institute of Electrical and Electronics Engineers Transactions on Neural Networks*, 4(5):737–739.

Bhattacharyya, D.K. & Kalita, J.K. 2013. *Network anomaly detection: a machine learning perspective*. London, United Kingdom: Chapman and Hall CRC.

Bhuyan, M.H., Bhattacharyya, D.K. & Kalita, J.K. 2014. Network anomaly detection: methods, systems and tools. *Institute of Electrical and Electronics Engineers Communications Surveys and Tutorials*, 16(1):303–336. doi:10.1109/SURV.2013.052213.00046.

Bhuyan, M.H., Bhattacharyya, D.K. & Kalita, J.K. 2015. Towards generating real-life datasets for network intrusion detection. *International Journal of Network Security*, 17(6):683–701. doi:10.5220/0006639801080116.

Bilge, L. & Dumitras, T. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. *Proceedings of the Conference on Computer and Communications Security, Raleigh, 16 October 2012*. New York, USA: Association for Computing Machinery: 833–844. doi:10.1145/2382196.2382284.

Black, P.E. & Bojanova, I. 2016. Defeating buffer overflow: a trivial but dangerous bug. *IT Professional*, 18(6):58–61. doi:10.1109/MITP.2016.117.

Boryczka, U. & Kozak, J. 2010. Ant colony decision trees a new method for constructing decision trees based on ant colony optimization. In J.-S. Pan, S.-M. Chen & N.T. Nguyen (eds.), *Proceedings of the International Conference on Computational Collective Intelligence, Kaohsiung, 10 November 2010*. New York, USA: Association for Computing Machinery: 373–382. doi:10.1007/978-3-642-16693-8.

Boryczka, U. & Kozak, J. 2011. An adaptive discretization in the ACDT algorithm for continuous attributes. *Lecture Notes in Computer Science*, 6923(2):475–484. doi:10.1007/978-3-642-23938-0 48.

Boryczka, U. & Kozak, J. 2012. Ant colony decision forest meta-ensemble. *Proceedings of the International Conference on Computational Collective Intelligence, Vietnam, 28 November 2012*. Heidelberg, Germany: Springer International: 473–482. doi:10.1007/978-3-642-34707-8 48. ISBN 9783642347061.

Boryczka, U., Probierz, B. & Kozak, J. 2016. Automatic categorization of email into folders by ant colony decision tree and social networks. In A. Caballero, R. Howlett & L. Jain (eds.), *Intelligent decision technologies*: 71–81. Heidelberg, Germany: Springer International.

Bouzida, Y. & Cuppens, F. 2006. Detecting known and novel network intrusions. S. Fischer-Hubner, K. Rannenberg, L. Yngstr¨ om & S. Lindskog (eds.), *Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection, Karlstad, 22 May 2006*, volume 201. Boston, USA: Springer International: 258–270. doi:10.1007/0-387-33406-8 22.

Bradley, A.P. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159. doi:10.1016/S0031-3203(96)00142-2.

Bramer, M. 2016. Using decision trees for classification. In *Principles of data mining*: 39–48. London, United Kingdom: Springer International.

Breiman, L. 1996. Bagging predictors. *Machine Learning*, 24(2):123–140. doi:10.1007/BF00058655.

Breiman, L., Friedman, J., Stone, C.J. & Olshen, R.A. 1984. *Classification and regression trees*. London, United Kingdom: Chapman and Hall CRC.

Brink, H., Richards, J. & Fetherolf, M. 2016. *Real-world machine learning*. Greenwich, United States: Manning Publications Co.

Buczak, A.L. & Guven, E. 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. *Institute of Electrical and Electronics Engineers Communications Surveys and Tutorials*, 18(2):1153–1176. doi:10.1109/COMST.2015.2494502.

Bukac, V., Tucek, P. & Deutsch, M. 2012. Advances and challenges in standalone host-based intrusion detection systems. In S. Fischer-Hubner, S. Katsikas & G. Quirchmayr (eds.), *Proceedings of the International Conference on Trust, Privacy and Security in Digital Business, Vienna, 3 September 2012*. Heidelberg, Germany: Springer International: 105–117. ISBN 978-3-642-32287-7.

Burdette, P. 2016. Timeline of an attack. *Network Security*, 2016(9):16–17. doi:10.1016/S1353-4858(16)30089-7.

Bursa, M. & Lhotska, L. 2015. Ant-inspired algorithms for decision tree induction. *Proceedings of the International Conference on Information Technology in Bio-and Medical Informatics, Valencia, 3 September 2015*. New York, USA: Springer International: 95–106. doi:10.1007/978-3-319-22741-2 9.

Bushev, A., Vlasenko, S., Glotov, I., Monakhov, Y. & Tishin, A. 2013. The development of the architecture of distributed network intrusion detection system (D-NIDS). *Computing Research Repository*, 1–3.

Canetti, R., Halevi, S. & Steiner, M. 2006. Mitigating dictionary attacks on password-protected local storage. In C. Dwork (ed.), *Advances in Cryptology*: 160–179. Heidelberg, Germany: Springer International.

Cannady, J.D. 1998. Artificial neural networks for misuse detection. *Proceedings of the National Information Systems Security Conference, Arlington, 5 October 1998*. Gaithersburg, USA: National Institute of Standards and Technology: 368–381.

Cecilia, J.M., Garcia, J.M., Ujaldon, M., Nisbet, A. & Amos, M. 2011. Parallelization strategies for ant colony optimisation on GPUs. *Proceedings of the International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, Shanghai, 16 May 2011*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 339–346. doi:10.1109/IPDPS.2011.170.

Chahal, J.K. & Kaur, A. 2016. Use of data mining techniques in intrusion detection: a survey. *Imperial Journal of Interdisciplinary Research*, 2(6):452–456. ISSN 2454-1362.
Chand, N., Mishra, P., Krishna, C.R., Pilli, E.S. & Govil, M.C. 2016. A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection. *Proceedings of the International Conference on Advances in Computing, Communication and Automation, Dehradun, 30 September 2016*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6. doi:10.1109/ICACCA.2016.7578859.

Chandola, V., Banerjee, A. & Kumar, V. 2009. Anomaly detection: a survey. *Association for Computing Machinery Computing Surveys*, 41(3):1–58. doi:10.1145/1541880.1541882.

Chauhan, P. & Chandra, N. 2013. A review on hybrid intrusion detection system using artificial immune system approaches. *International Journal of Computer Applications*, 68(20):22–27. doi:10.5120/11695-6499.

Chen, Z., Wei, P. & Delis, A. 2008. Catching remote administration trojans (RATs). *Software: Practice and Experience*, 38(7):667–703. doi:10.1002/spe.v38:7.

Chennupati, G. 2014. *eAnt-Miner : an ensemble ant-miner to improve the ACO classification*. https://arxiv.org/abs/1409.2710 [16/08/2017].

Clark, P.F. & Niblett, T. 1986. Learning {IF} {THEN} rules in noisy domains. In B. Phelps (ed.), *Interactions in artificial intelligence and statistical methods*: 84–97. Hampshire, England: Technical Press.

Correa, A., Nixon, A. & Bienkowski, T. 2016. *Investigating DDoS - architecture, actors, and attribution*. https://www.blackhat.com/docs/webcast/10202016-investigating-ddos.pdf [24/01/2017].

Cowan, C., Wagle, P., Pu, C., Beattie, S. & Walpole, J. 2003. Buffer overflows: attacks and defenses for the vulnerability of the decade. *Proceedings of the DARPA Information Survivability Conference and Exposition, Hilton Head, 25 January 2000*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 227–237. doi:10.1109/DISCEX.2000.821514.

Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M. & Karir, M. 2014. Taming the 800 pound gorilla: the rise and decline of NTP DDoS attacks. *Proceedings of the Conference on Internet Measurement, Vancouver, 5 November 2014*. New York, USA: Association for Computing Machinery: 435–448. doi:10.1145/2663716.2663717.

Damon, E., Dale, J., Laron, E., Mache, J., Land, N. & Weiss, R. 2012. Hands-on denial of service lab exercises using SlowLoris and RUDY. *Proceedings of the Information Security Curriculum Development Conference, Kennesaw, 12 October 2012*. New York, USA: Association for Computing Machinery: 21–29. doi:10.1145/2390317.2390321.

Darwish, M., Ouda, A. & Capretz, L.F. 2013. Cloud-based DDoS attacks and defenses. *Proceedings of the International Conference on Information Society, Toronto, 24 June 2013*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 67–71. ISBN 9781908320131.

De Vivo, M., Carrasco, E., Isern, G. & De Vivo, G.O. 1999. A review of port scanning techniques. *Association for Computing Machinery Sigcomm Computer Communication Review*, 29(2):41–48. doi:10.1145/505733.505737.

Debar, H., Becker, M. & Siboni, D. 1992. A neural network component for an intrusion detection system. *Proceedings of the Computer Society Symposium on Research in Security and Privacy, Oakland, 4 May 1992*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 240–250. doi:10.1109/RISP.1992.213257. ISBN 0-8186-2825-1.
Debar, H., Dacier, M. & Wespi, A. 1999. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822. doi:10.1016/S1389-1286(98)00017-6.

Debar, H., Dacier, M. & Wespi, A. 2000. A revised taxonomy for intrusion-detection systems. In *Annals of telecommunications*: 361–378. Heidelberg, Germany: Springer International.

Denning, D.E. 1987. An intrusion-detection model. *Institute of Electrical and Electronics Engineers Transactions on Software Engineering*, 13(2):222–232. doi:10.1109/TSE.1987.232894.

Depren, O., Topallar, M., Anarim, E. & Ciliz, M.K. 2005. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, 29(4):713–722. doi:10.1016/j.eswa.2005.05.002.

Devare, A., Shelake, M., Vahadne, V. & Tamboli, P.K.B. 2016. A system for denial-of-service attack detection based on multivariate correlation analysis. *Institute of Electrical and Electronics Engineers Transactions on Parallel and Distributed Systems*, 25(2):447–456. doi:10.1109/TPDS.2013.146.

Dey, P.K. 2016. *Prashant's algorithm for password management system*. *International Journal of Engineering Science and Computing,* 6(8):2424-2426.

Dietterich, T.G. 2000*a*. Ensemble methods in machine learning. In *Multiple classifier systems*: 1–15. Heidelberg, Germany: Springer International.

Dietterich, T.G. 2000*b*. An experimental comparison of three methods for constructing ensembles of decision trees. *Machine Learning*, 40(2):139–157. doi:10.1023/A:1007607513941.

Dinakaran, S. & Thangaiah, D.P.R.J. 2013. Role of attribute selection in classification algorithms. *International Journal of Scientific and Engineering Research*, 4(6):67–71. ISSN 2229-5518.

Domingos, P. 2012. A few useful things to know about machine learning. *Communications of the Association for Computing Machinery*, 55(10):78–87. doi:10.1145/2347736.2347755.

Dorigo, M., Birattari, M. & Stutzle, T. 2006. Ant colony optimization. *Institute of Electrical and Electronics Engineers Computational Intelligence*, 1(4):28–39. doi:10.1109/MCI.2006.329691.

Dorigo, M. & Gambardella, L.M. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Transactions on Evolutionary Computation*, 1(1):53–66. doi:10.1109/4235.585892.

Dorigo, M., Maniezzo, V. & Colorni, A. 1996. The ant systems: optimization by a colony of cooperative agents. *Institute of Electrical and Electronics Engineers Transactions on Man, Machine and Cybernetics*, 26(1):1–13. doi:10.1109/3477.484436.

Dumitru, C. & Maria, V. 2013. *Advantages and disadvantages of using neural networks for predictions*. http://stec.univovidius.ro/html/anale/ENG/cuprins%20rezumate/volum2013p1.pdf [16/08/2017].

Edwards, H. 2016. *A devastating type of hack is costing people big money*. http://time.com/4303129/hackers-computer-ransom-ransomware [19/01/2017].

Elomaa, T. 1994. In defense of C4.5 notes on learning one-level decision trees. In W.W. Cohen & H. Hirsh (eds.), *Proceedings of the International Conference on International Conference on Machine Learning, New Brunswick, 10 July 1994*. San Francisco, USA: Morgan Kaufmann Inc.: 62–69.

Engen, V. 2010. Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data. Ph.D. thesis, Bournemouth University, Poole.

Evangelista, P.F. 2005. Computer intrusion detection through statistical analysis and prediction modeling. Ph.D. thesis, Rensselaer Polytechnic Institute, New York.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874. doi:10.1016/j.patrec.2005.10.010.

Fayyad, U. & Irani, K. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambery, 1 September 1993*. California, United States: International Joint Conferences on Artificial Intelligence Inc.: 1022–1027.

Folino, G. & Sabatino, P. 2016. Ensemble based collaborative and distributed intrusion detection systems. *Journal of Network and Computer Applications*, 66:1–16. ISSN 1084-8045. doi:10.1016/j.jnca.2016.03.011.

Fontugne, R., Borgnat, P., Abry, P. & Fukuda, K. 2010. Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. *Proceedings of the International Conference on emerging Networking Experiments and Technologies, Philadelphia, 30 November 2010*. California, USA: The Association for Computing Machinery: 1–8.

Foster, B. 2015. *The true cost of false positives*. https://www.scmagazineuk.com/the-true-cost-of-false-positives/article/537128 [10/01/2017].

Frank, J. 1994. Artificial intelligence and intrusion detection: current and future directions. *Proceedings of the 17th National Computer Security Conference, Baltimore, 11 October 1994*. Gaithersburg, USA: National Institute of Standards and Technology. doi:10.1016/0167-4048(95)97010-8.

Frei, S. 2014. *FireEye: The known unknowns*. https://www.avantec.ch/assets/uploads/AVANTEC_The_Known_Unknowns_Stefan_Frei.pdf [24/11/2016].

Freund, Y. & Schapire, R.E. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. *Proceedings of the European conference on computational learning theory, Barcelona, 15 March 1995*. Heidelberg, Germany: Springer International: 23–37. doi:10.1007/3-540-59119-2 166.

Friedman, J.H. 1996. Lazy decision trees. *Proceedings of the National Conference on Artificial Intelligence, Portland, 4 August 1996*. California, USA: Association for the Advancement of Artificial Intelligence Press: 717–724.

Gambardella, L.M., Montemanni, R. & Weyland, D. 2012. Coupling ant colony systems with strong local searches. *European Journal of Operational Research*, 220(3):831–843. doi:10.1016/j.ejor.2012.02.038.

Gatta, R., Vallati, M., De Bari, B., Pasinetti, N., Cappelli, C., Pirola, I., Salvetti, M., Buglione, M., Muiesan, M.L. & Magrini, S. 2014. Information retrieval in medicine: an extensive experimental study. In A. Fred, C. Verdier, G. Plantier, H. Gamboa, M. Bienkiewicz & T. Schultz (eds.), *Proceedings of the International Conference on Health Informatics, Angers, 3 March 2014*. Setubal, Portugal: ScitePress. doi:10.5220/0004909904470452.

Geramiraz, F., Memaripour, A.S. & Abbaspour, M. 2012. Adaptive anomaly-based intrusion detection system using fuzzy controller. *International Journal of Network Security*, 14(6):352–361. ISSN 1816-3548.

Gollmann, D. 2010. Computer security. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(5):544–554. doi:10.1002/wics.106.

Goodman, L. 1999. Hypothesis-limited research. *Genome Research*, 9:673–676.

Gutzwiller, R.S. & Reeder, J. 2017. Human interactive machine learning for trust in teams of autonomous robots. *Proceedings of the Conference on Cognitive and Computational Aspects of Situation Management, Savannah, 27 March 2017*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–3. doi:10.1109/COGSIMA.2017.7929607.

Guzella, T.S. & Caminhas, W.M. 2009. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222. doi:10.1016/j.eswa.2009.02.037.

Gyanchandani, M., Rana, J.L. & Yadav, R.N. 2012. Taxonomy of anomaly based intrusion detection system: a review. *International Journal of Scientific and Research Publications*, 2(1):2250–3153. ISSN 2250-3153.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. 2009. The WEKA data mining software. *Association for Computing Machinery SIGKDD Explorations Newsletter*, 11(1):10. doi:10.1145/1656274.1656278.

Hall, M.A. & Holmes, G. 2003. Benchmarking attribute selection techniques for discrete class data mining. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*, 15(6):1437–1447. doi:10.1109/TKDE.2003.1245283.

Hamid, Y., Sugumaran, M. & Balasaraswathi, V.R. 2016. IDS using machine learning-current state of art and future directions. *British Journal of Applied Science and Technology*, 15(3):1–22. doi:10.9734/BJAST/2016/23668.

Han, S.J. & Cho, S.B. 2005. Evolutionary neural networks for anomaly detection based on the behavior of a program. *Institute of Electrical and Electronics Engineers Transactions on Systems, Man, and Cybernetics*, 36(3):559–570. doi:10.1109/TSMCB.2005.860136.

Hansen, L.K. & Salomon, P. 1990. Neural network ensembles. *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001. doi:10.1109/34.58871.

Harel, Y., Gal, I.B. & Elovici, Y. 2017. Cyber security and the role of intelligent systems in addressing its challenges. *Association for Computing Machinery Transactions on Intelligent Systems and Technology*, 8(4):1–12. doi:10.1145/3057729.

Hashmi, M.J., Saxena, M. & Saini, R. 2012. Classification of DDoS attacks and their defense techniques using intrusion prevention system. *International Journal of Computer Science and Communication Networks*, 2(5):607–614. ISSN 2249-5789.

Heater, B. 2016. *The growing threat of ransomware*. http://www.pcmag.com/news/343547/the-growing-threat-of-ransomware [24/01/2017].

Helal, A. & Otero, F.E.B. 2017. Automatic design of ant-miner mixed attributes for classification rule discovery. *Proceedings of the Genetic and Evolutionary Computation Conference, Berlin, 15 July 2017*. New York, USA: Association for Computing Machinery: 433–440. doi:10.1145/3071178.3071306.

Holden, G. 2004. *Guide to firewalls and network security: with intrusion detection and vpns*.
San Fransisco, United States: Course Technology. ISBN 9780619130398.

Holm, H. 2014. Signature based intrusion detection for zero-day attacks: (not) a closed chapter? *Proceedings of the Hawaii International Conference on System Sciences, Waikoloa, 6 January 2014*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 4895–4904. doi:10.1109/HICSS.2014.600.

Hoque, M.S., Mukit, M.A., Bikas, M.A.N. & Sazzadul Hoque, M. 2012. An implementation of intrusion detection system using genetic algorithm. *International Journal of Network Security Its Applications*, 4(2):109–120. doi:10.4156/jcit.vol4.issue1.janakiraman.

Hoque, N., Bhuyan, M.H., Baishya, R.C., Bhattacharyya, D.K. & Kalita, J.K. 2014. Network attacks: taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40(1):307–324. doi:10.1016/j.jnca.2013.08.001.

Hubballi, N. & Suryanarayanan, V. 2014. False alarm minimization techniques in signature-based intrusion detection systems: a survey. *Computer Communications*, 49:1–17. doi:10.1016/j.comcom.2014.04.012.

Inayat, Z., Gani, A., Anuar, N.B., Khan, M.K. & Anwar, S. 2016. Intrusion response systems: foundations, design, and challenges. *Journal of Network and Computer Applications*, 62:53–74. doi:10.1016/j.jnca.2015.12.006.

KDD Cup. 2016. *KDD Cup 99 data*. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [01/03/2016].

Kelly, J. 2006. An examination of pattern matching algorithms for intrusion detection systems. Master's thesis, Carleton University, Ottawa.

Kemmerer, R.A. & Vigna, G. 2002. Intrusion detection: a brief history and overview. *Institute of Electrical and Electronics Engineers Computer*, 35(4):27–30. doi:10.1109/MC.2002.10036.

Kendall, K.R. 1999. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, Cambridge.

Kent, C., Tanner, M. & Kabanda, S. 2016. How south african SMEs address cyber security: the case of web server logs and intrusion detection. *Proceedings of the International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies, Balaclava, 3 August 2016*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 100–105. doi:10.1109/EmergiTech.2016.7737319.

Kessler, G.C. 2000. *Defenses against distributed denial of service attacks*. https://www.giac.org/paper/gsec/236/defenses-distributed-denial-serviceattacks/100755 [16/08/2017].

Kevric, J., Jukic, S. & Subasi, A. 2016. An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*, 28(1):1051–1058. doi:10.1007/s00521-016-2418-1.
Kizza, J.M. 2017. Computer network security protocols. In *Guide to computer network security*: 365–396. New York, United States: Springer International.

Koch, R., Mario, G. & Rodosek, G. 2014. *Towards comparability of intrusion detection systems: new data sets*. https://tnc2014.terena.org/getfile/1388 [24/01/2017].

Kolias, C., Kambourakis, G. & Maragoudakis, M. 2011. Swarm intelligence in intrusion detection: a survey. *Computers & Security*, 30(8):625–642. doi:10.1016/j.cose.2011.08.009.

Korba, J. 2000. Windows NT attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, Cambridge.

Kotsiantis, S.B. 2007. Supervised machine learning: a review of classification techniques. *Informatica*, 31(3):249–268. doi:10.31449/inf.v31i3.148.

Kotsiantis, S.B. 2013. Decision trees: a recent overview. *Artificial Intelligence Review*, doi:10.1007/s10462-011-9272-4.

Kotsiantis, S.B., Kanellopoulos, D. & Pintelas, P.E. 2006. Data preprocessing for supervised learning. *International Journal of Computer Science*, 1(2):111–117. doi:10.1080/02331931003692557.

Kozak, J. & Boryczka, U. 2015. Multiple boosting in the ant colony decision forest meta-classifier. *Knowledge-based Systems*, 75:141–151. doi:10.1016/j.knosys.2014.11.027.

Kozak, J. & Boryczka, U. 2016. Collective data mining in the ant colony decision tree approach. *Information Sciences*, 372:126–147. doi:10.1016/j.ins.2016.08.051.

Kroll, J.A., Barocas, S., Felten, E.W., Reidenberg, J.R., Robinson, D.G. & Yu, H. 2016. *Accountable algorithms*. https://scholarship.law.upenn.edu/penn{_}law_review/vol165/iss3/3/ [16/08/2017].

Kruegel, C. & Toth, T. 2003. Using decision trees to improve signature-based intrusion detection. In G. Vigna, C. Kruegel & E. Jonsson (eds.), *Recent advances in intrusion detection*: 173–191. Heidelberg, Germany: Springer International.

Kulakow, S. 2001. *Netbus 2.1: is it still a trojan horse or an actual valid remote control administration tool*. https://www.sans.org/reading-room/whitepapers/malicious/ netbus-21-trojan-horse-actual-valid-remote-control-administration-tool-103 [16/08/2017].

Kumar, S. 2007. *Survey of current network intrusion detection techniques*. https://www.cse.wustl.edu/~jain/cse571-07/ids.htm [16/08/2017].

Kumar, S. & Sudarsan, S.D. 2014. An innovative UDP port scanning technique. *International Journal of Future Computer and Communication*, 3(6):381–384. doi:10.7763/IJFCC.2014.V3.332.

Lacy, S.E., Lones, M.A. & Smith, S.L. 2015. A comparison of evolved linear and non-linear ensemble vote aggregators. *Proceedings of the Congress on Evolutionary Computation, Sendai, 25 May 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers:
758–763. doi:10.1109/CEC.2015.7256967.

Larochelle, D. & Evans, D. 2001. Statically detecting likely buffer overflow vulnerabilities. *Proceedings of the Conference on USENIX Security Symposium, Washington, 13 August 2001*. Berkeley, USA: USENIX Association: 177–190.

Latkowski, R. 2003. High computational complexity of the decision tree induction with many missing attribute values. L. Czaja (ed.), *Proceedings of the International Workshop on Concurrency, Specification and Programming, Czarna, 25 September 2003*. Warsaw, Poland: Warsaw University: 318–325.

Lee, C., Roedel, C. & Silenok, E. 2003. *Detection and characterization of port scan attacks*. https://cseweb.ucsd.edu/~clbailey/PortScans.pdf [16/08/2017].

Lee, D., Carpenter, B.E. & Brownlee, N. 2010. Observations of UDP to TCP ratio and port numbers. *Proceedings of the International Conference on Internet Monitoring and Protection, Barcelona, 9 May 2010*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 99–104. doi:10.1109/ICIMP.2010.20.

Lee, W., Fan, W., Miller, M., Stolfo, S. & Zadok, E. 2002. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1):5–22. doi:10.3233/JCS-2002-101-202.

Lee, W. & Stolfo, S.J. 2000. A framework for constructing features and models for intrusion detection systems. *Association for Computing Machinery Transactions on Information and System Security*, 3(4):227–261. doi:10.1145/382912.382914.

Lee, W., Stolfo, S.J. & Mok, K.W. 1999. A data mining framework for building intrusion detection models. *Proceedings of the Institute of Electrical and Electronics Engineers Symposium on Security and Privacy, Oakland, 9 May 1999*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 120–132. doi:10.1109/SECPRI.1999.766909.

Liao, H.-J., Richard Lin, C.-H., Lin, Y.-C. & Tung, K.-Y. 2013. Intrusion detection system: a comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24. doi:10.1016/j.jnca.2012.09.004.

Lin, D. 2013. *Network intrusion detection and mitigation against denial of service attack*. https://repository.upenn.edu/cgi/viewcontent.cgi?article=2027&context=cisreports [16/08/2017].

Lin, Y., Zhang, Y. & Ou, Y.J. 2010. The design and implementation of host-based intrusion detection system. *Proceedings of the International Symposium on Intelligent Information Technology and Security Informatics, Jinggangshan, 2 April 2010*. New Jersey, USA:
Institute of Electrical and Electronics Engineers: 595–598. doi:10.1109/IITSI.2010.127.

Lindqvist, U. & Porras, P.A. 1999. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). *Proceedings of the Symposium on Security and Privacy, Oakland, 14 May 1999*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 146–161. doi:10.1109/SECPRI.1999.766911.

Ling, C.X., Yang, Q., Wang, J. & Zhang, S. 2004. Decision trees with minimal costs. *Proceedings of the International Conference on Machine learning, Banff, 4 July 2004*. New York, USA: Association for Computing Machinery: 69–72. doi:10.1145/1015330.1015369. ISBN 1581138285.

Lipitakis, A.-D. & Kotsiantis, S. 2015. Combining ensembles algorithms of symbolic learners.
Proceedings of the International Conference on Information, Intelligence, Systems and Applications, Corfu, 6 July 2015. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6. doi:10.1109/IISA.2015.7388118.

Lippmann, R., Haines, J.W., Fried, D.J., Korba, J. & Das, K. 2000. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595. doi:10.1016/S1389-1286(00)00139-0.

López-Ibáñez, M., Stützle, T. & Dorigo, M. 2017. Ant colony optimization: a component-wise overview. In *Handbook of heuristics*: 1–37. New York, United States: Springer International.

Lunt, T.F. 1990. IDES: an intelligent system for detecting intruders. *Proceedings of the Symposium: Computer Security, Threat and Countermeasures, Rome, 22 November 1990*.

Lyon, G.F. 2009. *Nmap network scanning: the official nmap project guide to network discovery and security scanning*. Sunnyvale, United States: Insecure.com LLC. ISBN 0979958717.

Maitra, S. & Madan, S. 2017. Intelligent cyber security solutions through high performance computing and data sciences: an integrated approach. *Institute of Information Technology and Management Journal of Management and IT*, 8(1):3–9. ISSN 0976-8629.

Małowidzki, M., Berezinski, P. & Mazur, M. 2015. Network intrusion detection: half a kingdom for a good dataset. *Proceedings of the North Atlantic Treaty Organization IST-SAS-139 Workshop, Lisbon, 21 April 2015*. Brussels, Belgium: NATO Science and Technology Organization: 1–6.

Maniezzo, V. 1999. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *Informs Journal on Computing*, 11(4):358–369. doi:10.1287/ijoc.11.4.358.

Maniezzo, V. 2002. Ant colony optimization: an overview. In *Essays and surveys in metaheuristics*: 469–492. New York, United States: Springer International.

Mavrovouniotis, M., Muller, F.M. & Yang, S. 2017. Ant colony optimization with local search for¨ dynamic traveling salesman problems. *Institute of Electrical and Electronics Engineers Transactions on Cybernetics*, 47(7):1743–1756. doi:10.1109/TCYB.2016.2556742.

Mavrovouniotis, M. & Yang, S. 2010. Ant colony optimization with direct communication for the traveling salesman problem. *Proceedings of the UK Workshop on Computational Intelligence, Colchester, 8 September 2010*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6. doi:10.1109/UKCI.2010.5625608.

McHugh, J. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *Association for Computing Machinery Transactions on Information and System Security*, 3(4):262–294. doi:10.1145/382912.382923.

McHugh, J., Christie, A. & Allen, J. 2000. Defending yourself: the role of intrusion detection systems. *Institute of Electrical and Electronics Engineers Software*, 17(5):42–51. doi:10.1109/52.877859.

Mirkovic, J., Benzel, T.V., Faber, T., Braden, R., Wroclawski, J.T. & Schwab, S. 2010. The DETER project: advancing the science of cyber security experimentation and test. *Proceedings of the International Conference on Technologies for Homeland Security, Waltham, 8 November 2010*. New Jersey, USA: Institute of Electrical and Electronics Engineers. doi:10.1109/THS.2010.5655108.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A. & Rajarajan, M. 2013. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57. doi:10.1016/j.jnca.2012.05.003.

Modi, C., Patel, D., Borisanya, B., Patel, A. & Rajarajan, M. 2012. A novel framework for intrusion detection in cloud. *Proceedings of the International Conference on Security of Information and Networks, Jaipur, 25 October 2012*. New York, USA: Association for Computing Machinery: 67–74. doi:10.1145/2388576.2388585.

Mohammad, M.N., Sulaiman, N. & Khalaf, E.T. 2011. A novel local network intrusion detection system based on support vector machine. *Journal of Computer Science*, 7(10):1560–1564. doi:10.3844/jcssp.2011.1560.1564.

Morrow, B. 2012. BYOD security challenges: control and protect your most sensitive data. *Network Security*, 2012(12):5–8. doi:10.1016/S1353-4858(12)70111-3.

Moustafa, N. & Slay, J. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems. *Proceedings of the Military Communications and Information Systems Conference, Canberra, 10 November 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6. doi:10.1109/MilCIS.2015.7348942.

Natesan, P. & Balasubramanie, P. 2012. Multi stage filter using enhanced adaboost for network intrusion detection. *International Journal of Network Security & Its Applications*, 4(3):121–135. doi:10.5121/ijnsa.2012.4308.

Nelissen, J. 2004. *Buffer overflows for dummies*. http://www.sans.org/rr/papers/60/481.pdf [16/08/2017].

Nguyen, H.A. & Choi, D. 2008. Application of data mining to network intrusion detection: classifier selection model. In *Challenges for next generation network operations and service management*: 399–408. Heidelberg, Germany: Springer International.

Niaki, S.T.A. & Moeinzadeh, B. 1997. A multivariate quality control procedure in multistage production systems. *International Journal of Engineering*, 10(4):191–208. doi:10.1080/00207540701504348.

Noureldien, N.A., Hussain, R.A. & Khalid, A. 2013. The effect of feature selection on detection accuracy of machine learning algorithms. *International Journal of Engineering Research and Technology*, 2(11):1407–1410. ISSN 2278-0181.

Noureldien, N.A. & Yousif, I.M. 2016. Accuracy of machine learning algorithms in detecting DoS attacks types. *Science and Technology*, 6(4):89–92. doi:10.5923/j.scit.20160604.01.

Ohta, S., Kurebayashi, R. & Kobayashi, K. 2008. Minimizing false positives of a decision tree classifier for intrusion detection on the internet. *Journal of Network and Systems Management*, 16(4):399–419. doi:10.1007/s10922-008-9102-4.

O'Leary, D.E. 1992. Intrusion-detection systems. *Journal of Information Systems*, 6(1):63–74. Omar, S., Ngadi, A. & Jebur, H.H. 2013. Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, 79(2):33–41. ISSN 0975-8887.

One, A. 1996. *Smashing the stack for fun and profit*. https://bit.ly/2TF7XP6 [11/08/2017].

Otero, F., Freitas, A. & Johnson, C. 2008. cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stutzle &W. A.F. (eds.), *Ant colony optimization and swarm intelligence*: 48–59. Heidelberg, Germany: Springer International.

Otero, F.E.B. 2016. *MYRA: an ACO framework for classification*. https://github.com/febo/myra [11/04/2016].

Otero, F.E.B., Freitas, A.A. & Johnson, C.G. 2012. Inducing decision trees with an ant colony optimization algorithm. *Applied Soft Computing Journal*, 12(11):3615–3626. doi:10.1016/j.asoc.2012.05.028.

Papernot, N., McDaniel, P., Sinha, A. & Wellman, M. 2016. *Towards the science of security and privacy in machine learning*. https://arxiv.org/pdf/1611.03814 [16/08/2017].

Parihar, L.S. & Tiwari, A. 2016. Survey on intrusion detection using data mining methods. *International Journal for Science and Advance Research in Technology*, 2(1):23–32. ISSN 2395-1052.

Parpinelli, R.S., Lopes, H.S. & Freitas, A.A. 2002*a*. An ant colony algorithm for classification rule discovery. In *Data mining: a heuristic approach*: 191–208. Hershey, United States: IDEA Group. ISBN 1930708254.

Parpinelli, R.S., Lopes, H.S. & Freitas, A.A. 2002*b*. Data mining with an ant colony optimization algorithm. *Institute of Electrical and Electronics Engineers Transactions on Evolutionary Computation*, 6(4):321–332. doi:10.1109/TEVC.2002.802452.

Patcha, A. & Park, J.-M. 2007. An overview of anomaly detection techniques: existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470. doi:10.1016/j.comnet.2007.02.001.

Patel, S.K. & Sonker, A. 2016. Rule-based network intrusion detection system for port scanning with efficient port scan detection rules using snort. *International Journal of Future Generation Communication and Networking*, 9(6):339–350. doi:10.14257/ijfgcn.2016.9.6.32.

Paulsen, C. 2016. Cybersecuring small businesses. *Institute of Electrical and Electronics Engineers Computer*, 49(8):92–97. doi:10.1109/MC.2016.223.

Peddabachigari, S., Abraham, A., Grosan, C. & Thomas, J. 2007. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114–132. doi:10.1016/j.jnca.2005.06.003.

Petersen, R. 2015. Data mining for network intrusion detection: a comparison of data mining algorithms and an analysis of relevant features for detecting cyber-attacks. Master's thesis, Mid Sweden University, Sundsvall.

Ponemon Institute. 2015. *The cost of malware containment*. https://bit.ly/2F8iOJc [21/02/2017].

Portnoy, L., Eskin, E. & Stolfo, S. 2001. Intrusion detection with unlabeled data using clustering. *Proceedings of the Workshop on Data Mining Applied to Security*. New York, USA: Association for Computing Machinery: 5–8.

Prakasam, A. & Savarimuthu, N. 2016. Metaheuristic algorithms and probabilistic behaviour: a comprehensive analysis of ant colony optimization and its variants. *Artificial Intelligence Review*, 45(1):97–130. doi:10.1007/s10462-015-9441-y.

Qayyum, A., Islam, M.H. & Jamil, M. 2005. Taxonomy of statistical based anomaly detection techniques for intrusion detection. *Proceedings of the Institute of Electrical and Electronics Engineers Symposium on Emerging Technologies, Islamabad, 18 September 2005*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 270–276. doi:10.1109/ICET.2005.1558893. ISBN 0780392477.

Quinlan, J.R. 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106. ISSN 15730565. doi:10.1023/A:1022643204877.

Quinlan, J.R. 1987. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234. doi:10.1016/S0020-7373(87)80053-6.

Quinlan, J.R. 1993. *C4.5 programs for machine learning*. San Mateo, United States: Morgan Kaufmann Inc. ISBN 1558602380.

Rai, K., Devi, M.S. & Guleria, A. 2016. Decision tree based algorithm for intrusion detection.
International Journal of Advanced Networking and Applications, 7(4):2828–2834. ISSN 0975-0290.

Raiyn, J. 2014. A survey of cyber attack detection strategies. *International Journal of Security and Its Applications*, 8(1):247–256. doi:10.14257/ijsia.2014.8.1.23.

Rajput, A.K., Tewani, R. & Dubey, A. 2016. The helping protocol DHCP. *Proceedings of the International Conference on Computing for Sustainable Global Development, New Delhi, 16 March 2016*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 634–637.

Rami, S.P. & Panchal, M.H. 2012. Comparative analysis of variations of ant-miner by varying input parameters. *International Journal of Computer Applications*, 60(3):25–29. doi:10.5120/9673-4097.

Ramos, V. & Abraham, A. 2005. ANTIDS: self-organized ant-based clustering model for intrusion detection system. In A. Abraham, Y. Dote, T. Furuhashi, M. Koppen, A. Ohuchi &¨ Y. Ohsawa (eds.), *Soft computing as transdisciplinary science and technology*: 977–986. Heidelberg, Germany: Springer International.

Randall, M. & Lewis, A. 2002. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*, 62(9):1421–1432. doi:10.1006/jpdc.2002.1854.

Rani, H. & Singh, J. 2017. Analysis of swarm intelligence optimization techniques used in MANETS: a survey. *International Journal of Advanced Research in Computer Science*, 8(5):636–639. ISSN 0976-5697.

Revathi, S. & Malathi, A. 2014. Detecting user-to-root (U2R) attacks based on various machine learning techniques. *International Journal of Advanced Research in Computer and Communication Engineering*, 3(4):6322–6324. ISSN 2319-5940.

Ristic, I. 2005. *Apache security*. Sebastopol, United States: O'Reilly Media Inc. ISBN 0596007248.

Rittinghouse, J. & Hancock, W.M. 2003. *Cybersecurity operations handbook*. Burlington, United States: Digital Press. ISBN 1555583067.

Rokach, L. & Maimon, O. 2005. Decision trees. In R.L. Maimon O. (ed.), *Data mining and knowledge discovery handbook*: 165–192. Boston, United States: Springer International.

Rokach, L. & Maimon, O. 2015. Data mining with decision trees: theory and applications. *Online Information Review*, 39(3):437–438. doi:10.1108/OIR-04-2015-0121.

Rudd, E., Rozsa, A., Gunther, M. & Boult, T. 2017. A survey of stealth malware: attacks, mitigation measures, and steps toward autonomous open world solutions. *Institute of Electrical and Electronics Engineers Communications Surveys and Tutorials*, 19(2):1145–1172. doi:10.1109/COMST.2016.2636078.

Russell, I. & Markov, Z. 2006. An introduction to the weka data mining system. *Proceedings of the Special Interest Group on Computer Science Education, Bologna, 26 June 2006*. New York, USA: Association for Computing Machinery: 742–742. doi:10.1145/1140123.1140127.

Rutkowski, L., Jaworski, M., Pietruczuk, L. & Duda, P. 2014. Decision trees for mining data streams based on the gaussian approximation. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*, 26(1):108–119. doi:10.1109/TKDE.2013.34.

Sabhnani, M. & Serpen, G. 2003. KDD feature set complaint heuristic rules for R2L attack detection. *Proceedings of the International Conference on Security and Management, Las Vegas, 23 June 2003*. USA: CSREA Press: 310–316. ISBN 1932415165.

Sadeghian, A., Zamani, M. & Ibrahim, S. 2013. SQL injection is still alive: a study on SQL injection signature evasion techniques. *Proceedings of the International Conference on Informatics and Creative Multimedia, Kuala Lumpur, 4 September 2013*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 265–268. doi:10.1109/ICICM.2013.52.

Sahasrabuddhe, A., Naikade, S., Ramaswamy, A., Sadliwala, B. & Futane, P. 2017. Survey on intrusion detection system using data mining techniques. *International Research Journal of Engineering and Technology*, 4(5):1780–1784. ISSN 2395-0072.

Salama, K.M., Abdelbar, A.M. & Otero, F.E.B. 2016. Investigating evaluation measures in ant colony algorithms for learning decision tree classifiers. *Proceedings of the Institute of Electrical and Electronics Engineers Symposium Series on Computational Intelligence, Cape Town, 7 December 2015*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1146–1153. doi:10.1109/SSCI.2015.164.

Salama, K.M. & Otero, F.E.B. 2014. Learning multi-tree classification models with ant colony optimization. *Proceedings of the International Conference on Evolutionary Computation Theory and Applications, Rome, 22 October 2014*. Setubal, Portugal: Institute for Systems and Technologies of Information, Control and Communication Press: 38–48.

Scarfone, K.A. & Mell, P.M. 2007. *Guide to intrusion detection and prevention systems*. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf [16/08/2017].

Schapire, R. 2015. *Machine learning algorithms for classification*. https://www.cs.princeton.edu/~schapire/talks/picasso-minicourse.pdf [24/01/2017].

Schmidhuber, J. 2015. Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117. doi:10.1016/j.neunet.2014.09.003.

Schupp, S. 2000. *Limitation of network intrusion detection*. https://bit.ly/2Cca82H [16/08/2017].

Shackelford, S.J. 2016. Business and cyber peace: we need you! *Business Horizons*, 59(5):539–548. doi:10.1016/j.bushor.2016.03.015.

Sharma, P. & Kunwar, R.S. 2016. Cyber attacks on intrusion detection system. *International Journal of Information*, 6(2):191–196. doi:10.5121/ijist.2016.6220.

Shiravi, A., Shiravi, H., Tavallaee, M. & Ghorbani, A.A. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374. doi:10.1016/j.cose.2011.12.012.

Shrivastava, N. & Richariya, V. 2012. Ant colony optimization with classification algorithms used for intrusion detection. *International Journal of Computational Engineering and Management*, 15(1):54–64. ISSN 2230-7893.

Singh, J. & Nene, M.J. 2013. A survey on machine learning techniques for intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(11):4349–4355. ISSN 2319-5940.

Singh, R.R. & Tomar, D. 2015. Network forensics: detection and analysis of stealth port scanning attack. *International Journal of Computer Networks and Communications Security*, 3(2):33–42. ISSN 2410-0595.

Somal, I.K. & Virk, S. 2014. Classification of distributed denial of service attacks–architecture, taxonomy and tools. *International Journal of Advanced Research in Computer Science and Technology*, 2(2):118–122. ISSN 2347-9817.

Sommer, R. & Paxson, V. 2010. Outside the closed world: on using machine learning for network intrusion detection. *Proceedings of the Institute of Electrical and Electronics Engineers Symposium on Security and Privacy, Oakland, 16 May 2010*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 305–316. doi:10.1109/SP.2010.25.

Soroush, E., Abadeh, M.S. & Habibi, J. 2006. A boosting ant-colony optimization algorithm for computer intrusion detection. *Proceedings of the International Symposium on Frontiers in Networking with Applications, Vienna, 18 April 2006*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 1–6.

Staudemeyer, R.C. & Omlin, C.W. 2014. Extracting salient features for network intrusion detection using machine learning methods. *South African Computer Journal*, 52:82–96. doi:10.18489/sacj.v52i0.200.

Stewart, J.M. 2013. *Network security, firewalls and VPNs*. Burlington, United States: Jones and Bartlett Learning.

Stiawan, D., Idris, M.Y.B., Abdullah, A.H., AlQurashi, M. & Budiarto, R. 2016. Penetration testing and mitigation of vulnerabilities windows server. *International Journal of Network Security*, 18(3):501–513. doi:10.6633/IJNS.

Stingley, M. 2015. *Infrastructure security architecture: effective security monitoring*. https://bit.ly/2O2QhYd [16/08/2017].

Stutzle, T. & Hoos, H. 1998. Improvements on the ant-system: introducing the MAX-MIN ant system. In *Artificial neural nets and genetic algorithms*: 245–249. New York, United States: Springer International.

Stutzle, T. & Hoos, H.H. 2000. MAX-MIN ant system. *Future Generation Computer Systems*, 16(8):889–914. doi:10.1016/S0167-739X(00)00043-1.

Kumar, G.S. 2012. Real time and offline network intrusion detection using improved decision tree algorithm. *International Journal of Computer Applications*, 48(25):1–6. doi:10.5120/7541-0482.

Surjandari, I., Dhini, A., Rachman, A. & Novita, R. 2015. Estimation of dry docking duration using a numerical ant colony decision tree. *International Journal of Applied Management Science*, 7(2):164–175. doi:10.1504/IJAMS.2015.069264.

Tan, Z., Jamdagni, A., He, X., Nanda, P. & Liu, R.P. 2014. A system for denial-of-service attack detection based on multivariate correlation analysis. *Institute of Electrical and Electronics Engineers Transactions on Parallel and Distributed Systems*, 25(2):447–456. doi:10.1109/TPDS.2013.146.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A.-A. 2009. A detailed analysis of the KDD CUP 99 data set. *Proceedings of the Symposium on Computational Intelligence for Security and Defence Applications, Ottawa, 8 July 2009*. New Jersey, USA: Institute of Electrical and Electronics Engineers. doi:10.1109/CISDA.2009.5356528.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A.-A. 2011. *NSL-KDD data set*. http://nsl.cs.unb.ca/NSL-KDD/ [24/03/2017].

Tavallaee, M., Stakhanova, N. & Ghorbani, A.A. 2010. Toward credible evaluation of anomaly-based intrusion-detection methods. *Institute of Electrical and Electronics Engineers Transactions on Systems, Man, and Cybernetics*, 40(5):516–524. doi:10.1109/TSMCC.2010.2048428.

Team, C. *et al.*. 2009. *The cooperative association for internet data analysis*. https://www.caida.org/data/about/downloads/tables.xml [24/01/2017].

Tesfahun, A. & Lalitha Bhaskari, D. 2013. Intrusion detection using random forests classifier with SMOTE and feature reduction. *Proceedings of the International Conference on Cloud, Ubiquitous Computing and Emerging Technologies, Pune, 15 November 2013*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 127–132. doi:10.1109/CUBE.2013.31.

The Economist Intelligence Unit. 2016. *The year in crisis: evolution of risk*. https://bit.ly/2VULz1A [24/01/2017].

Thomas, C., Sharma, V. & Balakrishnan, N. 2008. Usefulness of DARPA dataset for intrusion detection system evaluation. W.R. William J. Tolone (ed.), *Proceedings of the SPIE Defense and Security Symposium, Orlando, 16 March 2008*. Washington, USA: The International Society for Optical Engineering. doi:10.1117/12.777341.

Tsai, C.F., Hsu, Y.F., Lin, C.Y. & Lin, W.Y. 2009. Intrusion detection by machine learning: a review. *Expert Systems with Applications*, 36(10):11994–12000. doi:10.1016/j.eswa.2009.05.029.

Tsang, C.-H. & Kwong, S. 2006. Ant colony clustering and feature extraction for anomaly intrusion detection. In A. Abraham, C. Grosan & V. Ramos (eds.), *Swarm intelligence in data mining*: 101–123. Heidelberg, Germany: Springer International.

Tupakula, U., Varadharajan, V. & Akku, N. 2011. Intrusion detection techniques for infrastructure as a service cloud. *Proceedings of the International Conference on Dependable, Autonomic and Secure Computing, Sydney, 11 December 2011*, 9. New Jersey, USA: Institute of Electrical and Electronics Engineers: 744–751. doi:10.1109/DASC.2011.128.

Van Rijswijk-Deij, R., Sperotto, A. & Pras, A. 2014. DNSSEC and its potential for DDoS attacks. *Proceedings of the Conference on Internet Measurement Conference, Vancouver, 5 November 2014*. New York, USA: Association for Computing Machinery: 449–460. doi:10.1145/2663716.2663731.

Van Trees, H.L. 2002. *Classical detection and estimation theory*. New Jersey, United States: John Wiley & Sons.

Vasilomanolakis, E., Stahn, M., Cordero, C.G. & Muhlhauser, M. 2016. On probe-response attacks in collaborative intrusion detection systems. *Proceedings of the Conference on Communications and Network Security, Philadelphia, 17 October 2016*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 279–286. doi:10.1109/CNS.2016.7860495.

Venkataramanan, R., Jeong, M.-W. & Balaji, P. 2011. A flow and packet level model of the internet. *Yale University Department of Electrical Engineering*, 1–8.

Verwoerd, T. & Hunt, R. 2002. Intrusion detection techniques and approaches. *Computer Communications*, 25(15):1356–1365. doi:10.1016/S0140-3664(02)00037-3.

Vigna, G. & Kruegel, C. 2005. *Host-based intrusion detection*. https://www.cs.ucsb.edu/~chris/research/doc/infsec05_hids.pdf [16/08/2017].

Von Solms, R. & Van Niekerk, J. 2013. From information security to cyber security. *Computers and Security*, 38:97–102. doi:10.1016/j.cose.2013.04.004.

Wahba, Y., Elsalamouny, E. & Eltaweel, G. 2015. Improving the performance of multi-class intrusion detection systems using feature reduction. *International Journal of Computer Science Issues*, 12(3):255–262. ISSN 1694-0814.

Wang, P.-Y. & Hong, M.-Q. 2016. A secure management scheme designed in cloud. *Proceedings of the International Conference on High Performance and Smart Computing (HPSC), New York, 08 April 2016*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 158–162.

Wang, Y. 2008. *Statistical techniques for network security: modern statistically-based intrusion detection and protection*. Hersey, United Stated: IGI Global. ISBN 159904708X.

Wange, S., Sahu, S.K. & Mishra, A. 2016. A critical analysis on intrusion detection techniques. *The Association of Computer, Communication and Education for National Triumph Social and Welfare Society*, 3:77–81. doi:10.19101/IJATEE.2016.319001.

Weng, C.G. & Poon, J. 2008. A new evaluation measure for imbalanced datasets. In J.F. Roddick, J. Li, P. Christen & P. Kennedy (eds.), *Proceedings of the Australasian Data Mining Conference, Glenelg, 27 November 2018*. Darlinghurst, Australia: Australian Computer Society, Inc.: 27–32. ISBN 9781920682682. ISSN 14451336.

Wilhelm, T. 2009. *Professional penetration testing: creating and operating a formal hacking lab*. Waltham, United States: Syngress. ISBN 1597499935.

Williams, J. 2016. *The true cost of false positives in application security*. https://bit.ly/2CheCVU [16/02/2017].

Witten, I., Frank, E., Hall, M. & Pal, C.J. 2016. *Data mining: practical machine learning tools and techniques*. Cambridge, United States: Morgan Kaufmann. ISBN 9780128042915.

Wu, S.X. & Banzhaf, W. 2010. The use of computational intelligence in intrusion detection systems: a review. *Applied Soft Computing*, 10(1):1–35. doi:10.1016/j.asoc.2009.06.019.

Wueest, C. 2014. *The continued rise of DDoS attacks*. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-continued-rise-of-ddos-attacks.pdf [16/08/2017].

Yang, Z.R. 2010. *Machine learning approaches to bioinformatics*. New Jersey, United States: World Scientific Co.

Ye, N. 2000. A markov chain model of temporal behavior for anomaly detection. *Proceedings of the Institute of Electrical and Electronics Engineers Workshop on Information Assurance and Security, New York, 6 June 2000*. New Jersey, USA: Institute of Electrical and Electronics Engineers: 171–174.

Yegneswaran, V., Giffin, J.T., Barford, P. & Jha, S. 2005. An architecture for generating semantic aware signatures. *Proceedings of the USENIX Security Symposium, Baltimore, 31 July 2005*. California, USA: USENIX Association Berkeley: 97–112.

Yost, J.R. 2016. The march of IDES: early history of intrusion-detection expert systems. *Institute of Electrical and Electronics Engineers Annals of the History of Computing*, 38(4):42–54. doi:10.1109/MAHC.2015.41.

Zetter, K. 2014. *Countdown to zero day: stuxnet and the launch of the world's first digital weapon*. New York, United States: Crown Group. ISBN 0770436196.

Zhu, L., Hu, Z., Heidemann, J., Wessels, D., Mankin, A. & Somaiya, N. 2015. T-DNS: connection-oriented DNS to improve privacy and security. *Association for Computing Machinery's Special Interest Group on Data Communications*, 44(4):379–380. doi:10.1145/2740070.2631442.

APPENDIX A

FEATURE SELECTION RESULTS

A.1 Probe Weka output

=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 42 xAttack):
Information Gain Ranking Filter

Ranked attributes:
0.4680165 2 src_bytes
0.4348688 40 service
0.2986306 3 dst_bytes
0.2226567 30 dst_host_srv_count
0.218387 34 dst_host_srv_diff_host_rate
0.2145876 9 logged_in
0.2029187 33 dst_host_same_src_port_rate
0.1830301 32 dst_host_diff_srv_rate
0.1630371 31 dst_host_same_srv_rate
0.1545942 38 dst_host_srv_rerror_rate
0.1427631 37 dst_host_rerror_rate
0.140313 41 flag
0.1379103 24 rerror_rate
0.1148867 25 srv_rerror_rate
0.1109277 29 dst_host_count
0.1076456 39 protocol_type
0.1005102 20 count
0.0992447 26 same_srv_rate
0.094842 28 srv_diff_host_rate
0.0846589 27 diff_srv_rate
0.0579857 21 srv_count
0.0545715 22 serror_rate
0.0503095 35 dst_host_serror_rate
0.0296329 1 duration
0.025486 36 dst_host_srv_serror_rate
0.0091222 23 srv_serror_rate*
0.0027702 7 hot*
0.0020628 19 is_guest_login*

0.0015192 13 num_root*
0.0010834 16 num_access_files*
0.0009723 10 num_compromised*
0.0004327 11 root_shell*
0.0003028 12 su_attempted*
0.0000144 4 land*
0 18 is_host_login*
0 15 num_shells*
0 5 wrong_fragment*
0 6 urgent*
0 14 num_file_creations*
0 8 num_failed_logins*
0 17 num_outbound_cmds*

Selected attributes:
2,40,3,30,34,9,33,32,31,38,37,41,24,25,29,39,20,26,28,27,21,22,35,1,36 : 26
** - feature removed*

A.2 U2R Weka output

=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 42 xAttack):
Information Gain Ranking Filter

Ranked attributes:
0.0030552583 11 root_shell
0.0029812954 40 service
0.002837755 7 hot
0.0026270238 30 dst_host_srv_count
0.0023905689 10 num_compromised
0.0020280154 29 dst_host_count
0.0015656604 14 num_file_creations
0.0014369499 1 duration
0.0013610344 33 dst_host_same_src_port_rate
0.001017264 21 srv_count
0.0007670554 6 urgent
0.0006807885 13 num_root
0.000141594 9 logged_in
0.0000709713 41 flag
0.0000678283 39 protocol_type
0.0000149033 19 is_guest_login
0.0000000876 4 land
0 38 dst_host_srv_rerror_rate*

0 12 su_attempted*
0 37 dst_host_rerror_rate*
0 8 num_failed_logins*
0 15 num_shells*
0 2 src_bytes*
0 3 dst_bytes*
0 5 wrong_fragment*
0 36 dst_host_srv_serror_rate*
0 31 dst_host_same_srv_rate*
0 32 dst_host_diff_srv_rate*
0 27 diff_srv_rate*
0 26 same_srv_rate*
0 28 srv_diff_host_rate*
0 17 num_outbound_cmds*
0 34 dst_host_srv_diff_host_rate*
0 25 srv_rerror_rate*
0 24 rerror_rate*
0 23 srv_serror_rate*
0 22 serror_rate*
0 18 is_host_login*
0 20 count*
0 35 dst_host_serror_rate*
0 16 num_access_files*

Selected attributes: 11,40,7,30,10,29,14,1,33,21,6,13,9,41,39,19,4 : 18
** - feature removed*

A.3 R2L Weka output

=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 42 xAttack):
Information Gain Ranking Filter

Ranked attributes:
0.06627103 2 src_bytes
0.0485375 40 service
0.04114705 3 dst_bytes
0.03155934 30 dst_host_srv_count
0.02995786 7 hot
0.02988673 33 dst_host_same_src_port_rate
0.02309435 34 dst_host_srv_diff_host_rate
0.01939977 21 srv_count
0.01741531 20 count

0.01698704 29 dst_host_count
0.01377394 19 is_guest_login
0.0075492 28 srv_diff_host_rate
0.0069388 32 dst_host_diff_srv_rate
0.00504337 39 protocol_type
0.00445705 1 duration
0.00438698 36 dst_host_srv_serror_rate
0.00348874 31 dst_host_same_srv_rate
0.00330218 41 flag
0.00285783 8 num_failed_logins
0.00220367 9 logged_in
0.00202793 35 dst_host_serror_rate
0.00073021 27 diff_srv_rate
0.0003131 11 root_shell
0.00000163 4 land
0 6 urgent*
0 38 dst_host_srv_rerror_rate*
0 37 dst_host_rerror_rate*
0 5 wrong_fragment*
0 10 num_compromised*
0 26 same_srv_rate*
0 12 su_attempted*
0 23 srv_serror_rate*
0 24 rerror_rate*
0 18 is_host_login*
0 22 serror_rate*
0 17 num_outbound_cmds*
0 25 srv_rerror_rate*
0 14 num_file_creations*
0 15 num_shells*
0 16 num_access_files*
0 13 num_root*

Selected attributes: 2,40,3,30,7,33,34,21,20,29,19,28,32,39,1,36,31,41,8,9,35,27,11,4 : 24
** - feature removed*

A.4 DOS Weka output

=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 42 xAttack):
Information Gain Ranking Filter

Ranked attributes: 0.82137957 2 src_bytes

0.70151747 27 diff_srv_rate
0.67011644 26 same_srv_rate
0.65622599 41 flag
0.65225583 40 service
0.62349862 3 dst_bytes
0.56179273 20 count
0.54621635 36 dst_host_srv_serror_rate
0.54490639 35 dst_host_serror_rate
0.54021198 31 dst_host_same_srv_rate
0.53594495 32 dst_host_diff_srv_rate
0.52902426 22 serror_rate
0.52835117 23 srv_serror_rate
0.50604478 30 dst_host_srv_count
0.40926381 9 logged_in
0.25300156 34 dst_host_srv_diff_host_rate
0.24004231 29 dst_host_count
0.20397168 33 dst_host_same_src_port_rate
0.1741022 21 srv_count
0.16670613 28 srv_diff_host_rate
0.06340499 38 dst_host_srv_rerror_rate
0.06213791 39 protocol_type
0.05329209 1 duration
0.04211447 37 dst_host_rerror_rate
0.03001073 24 rerror_rate
0.02765923 25 srv_rerror_rate
0.01290771 5 wrong_fragment*
0.01206229 7 hot*
0.00901577 10 num_compromised*
0.0056462 19 is_guest_login*
0.00433829 13 num_root*
0.00249886 16 num_access_files*
0.00186479 14 num_file_creations*
0.00099824 11 root_shell*
0.00069858 12 su_attempted*
0.00039909 8 num_failed_logins*
0.00026603 15 num_shells*
0.00000224 4 land*
0 18 is_host_login*
0 6 urgent*
0 17 num_outbound_cmds*

Selected attributes:
2,27,26,41,40,3,20,36,35,31,32,22,23,30,9,34,29,33,21,28,38,39,1,37,24,25 : 26
** - feature removed*

A.5 20Train Weka output

=== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 42 xAttack):
Information Gain Ranking Filter

Ranked attributes:
0.806777083 2 src_bytes
0.632370663 40 service
0.631947382 3 dst_bytes
0.519250146 41 flag
0.515902949 27 diff_srv_rate
0.507754237 26 same_srv_rate
0.47284563 30 dst_host_srv_count
0.43902981 31 dst_host_same_srv_rate
0.412562311 32 dst_host_diff_srv_rate
0.403611513 35 dst_host_serror_rate
0.401731617 9 logged_in
0.396138161 36 dst_host_srv_serror_rate
0.390504636 22 serror_rate
0.382406912 20 count
0.377279134 23 srv_serror_rate
0.268769398 34 dst_host_srv_diff_host_rate
0.194957487 29 dst_host_count
0.192532029 33 dst_host_same_src_port_rate
0.144081691 28 srv_diff_host_rate
0.093588834 21 srv_count
0.088691095 38 dst_host_srv_rerror_rate
0.063682209 39 protocol_type
0.057011737 24 rerror_rate
0.053991071 37 dst_host_rerror_rate
0.052479105 25 srv_rerror_rate
0.034316178 1 duration
0.011069506 7 hot*
0.009857171 5 wrong_fragment*
0.006294468 10 num_compromised*
0.003833062 13 num_root*
0.001968948 16 num_access_files*
0.001131461 19 is_guest_login*
0.000844263 14 num_file_creations*
0.000755233 12 su_attempted*
0.000265543 11 root_shell*
0.000151506 15 num_shells*
0.000000263 4 land*
0 6 urgent*
0 17 num_outbound_cmds*
0 18 is_host_login*
0 8 num_failed_logins*

Selected attributes:

2,40,3,41,27,26,30,31,32,35,9,36,22,20,23,34,29,33,28,21,38,39,24,37,25,1 : 26

** - feature removed*

APPENDIX B

EXPERIMENT CYCLE TWO – COMBINED RESULT

Evaluation	20Train	20Train -FS	U2R	U2R -FS	R2L	R2L -FS	Probe	Probe -FS	DOS	DOS -FS	ATMa
TP	1828	1810	2149	2150	2114	2113	1911	1942	2010	1974	6787
FP	4383	4287	185	193	2518	2490	760	821	1480	1438	350
TN	5315	5411	17	9	236	264	1642	1582	2860	2903	1803
FN	324	342	3	3	38	39	241	210	142	179	2911
Error Rate (%)	39.72	39.06	7.99	8.31	52.1	51.56	21.99	22.63	24.99	24.89	27.52
Accuracy (%)	60.28	60.94	92.01	91.69	47.9	48.44	78.01	77.37	75.01	75.11	72.48
DR (%)	84.94	84.11	99.86	99.88	98.22	98.17	88.79	90.24	93	92	69.98
FAR (%)	45	44	92	96	91	90	32	34	34	33	16
Tree Quality (%)	98.91	99	99.91	99.91	99.55	99.56	99.39	99.42	99.72	99.69	98.97
Total Nodes	239.4	196.9	5	6	49	41.8	84	130.2	98.6	111.2	308
F-measure	0.44	0.44	0.96	0.96	0.62	0.63	0.79	0.79	0.71	0.7	0.81
Leaf Nodes	207.1	158.2	3	3.5	35.2	28.2	63.5	107.4	83.1	96.2	275.67
Runtime (s)	241.46	239.09	2.04	1.74	20.1	20.35	61.45	59.75	157.88	134.27	662.55
Tree Instances	143	140.9	10.5	55.8	106.9	122.8	126.1	133.3	138.6	136.7	145
Cost	852.67	875.73	17.69	17.29	114.51	110.65	303.34	347.52	1819.41	1419.46	2621.57

APPENDIX C

EXAMPLE DECISION TREES AND PREDICTION MODEL

This appendix includes examples of decision trees extracted during the experimentation process. Section C.2 illustrates a prediction model and a short explanation of how to interpret the model.

C.1 ATM example decision tree

```
'src_bytes' <= 28.0:
|   'same_srv_rate' <= 0.49:
|   |   'dst_bytes' <= 0.0: 1 (8717.0/5.0)
|   |   'dst_bytes' > 0.0:
|   |   |   'dst_host_srv_count' <= 1.0: 1 (11.0)
|   |   |   'dst_host_srv_count' > 1.0:
|   |   |   |   'count' <= 7.0: 0 (5.0)
|   |   |   |   'count' > 7.0: 1 (9.0)
|   'same_srv_rate' > 0.49:
|   |   'src_bytes' <= 0.0:
|   |   |   'dst_host_serror_rate' <= 0.76:
|   |   |   |   'dst_host_same_srv_rate' <= 0.18:
|   |   |   |   |   'dst_host_count' > 238.0: 1 (210.0/7.0)
|   |   |   |   |   'dst_host_count' <= 238.0:
|   |   |   |   |   |   'dst_host_diff_srv_rate' > 0.12: 1 (159.0/2.0)
|   |   |   |   |   |   'dst_host_diff_srv_rate' <= 0.12:
|   |   |   |   |   |   |   'dst_host_diff_srv_rate' <= 0.04: 0 (12.0)
|   |   |   |   |   |   |   'dst_host_diff_srv_rate' > 0.04: 1 (18.0/6.0)
|   |   |   |   'dst_host_same_srv_rate' > 0.18:
|   |   |   |   |   'dst_host_srv_count' > 17.0: 0 (611.0/8.0)
|   |   |   |   |   'dst_host_srv_count' <= 17.0:
|   |   |   |   |   |   'service' = 1: 0 (0.0)
|   |   |   |   |   |   'service' = 2: 0 (0.0)
|   |   |   |   |   |   'service' = 3: 0 (0.0)
|   |   |   |   |   |   'service' = 4: 0 (0.0)
|   |   |   |   |   |   'service' = 5: 0 (0.0)
|   |   |   |   |   |   'service' = 6: 0 (0.0)
|   |   |   |   |   |   'service' = 7: 0 (0.0)
|   |   |   |   |   |   'service' = 8: 0 (0.0)
```

```
|   |   |   |   |   |   'service' = 9: 0 (0.0)
|   |   |   |   |   |   'service' = 10: 0 (0.0)
|   |   |   |   |   |   'service' = 11: 0 (0.0)
|   |   |   |   |   |   'service' = 12: 0 (0.0)
|   |   |   |   |   |   'service' = 13: 0 (0.0)
|   |   |   |   |   |   'service' = 14: 0 (0.0)
|   |   |   |   |   |   'service' = 15: 0 (0.0)
|   |   |   |   |   |   'service' = 16: 0 (0.0)
|   |   |   |   |   |   'service' = 17: 0 (0.0)
|   |   |   |   |   |   'service' = 18: 0 (0.0)
|   |   |   |   |   |   'service' = 19: 0 (0.0)
|   |   |   |   |   |   'service' = 20: 1 (12.0)
|   |   |   |   |   |   'service' = 21: 0 (0.0)
|   |   |   |   |   |   'service' = 22: 0 (0.0)
|   |   |   |   |   |   'service' = 23: 0 (0.0)
|   |   |   |   |   |   'service' = 24: 0 (0.0)
|   |   |   |   |   |   'service' = 25: 0 (16.0/1.0)
|   |   |   |   |   |   'service' = 26: 0 (0.0)
|   |   |   |   |   |   'service' = 27: 0 (0.0)
|   |   |   |   |   |   'service' = 28: 0 (0.0)
|   |   |   |   |   |   'service' = 29: 1 (3.0)
|   |   |   |   |   |   'service' = 30: 0 (0.0)
|   |   |   |   |   |   'service' = 31: 0 (0.0)
|   |   |   |   |   |   'service' = 32: 0 (0.0)
|   |   |   |   |   |   'service' = 33: 0 (0.0)
|   |   |   |   |   |   'service' = 34: 0 (0.0)
|   |   |   |   |   |   'service' = 35: 0 (0.0)
|   |   |   |   |   |   'service' = 36: 0 (0.0)
|   |   |   |   |   |   'service' = 37: 0 (0.0)
|   |   |   |   |   |   'service' = 38: 0 (0.0)
|   |   |   |   |   |   'service' = 39: 0 (0.0)
|   |   |   |   |   |   'service' = 40: 0 (0.0)
|   |   |   |   |   |   'service' = 41: 0 (0.0)
|   |   |   |   |   |   'service' = 42: 0 (0.0)
|   |   |   |   |   |   'service' = 43: 0 (0.0)
|   |   |   |   |   |   'service' = 44: 0 (0.0)
|   |   |   |   |   |   'service' = 45: 0 (3.0)
|   |   |   |   |   |   'service' = 46: 0 (0.0)
|   |   |   |   |   |   'service' = 47: 0 (0.0)
|   |   |   |   |   |   'service' = 48: 0 (0.0)
|   |   |   |   |   |   'service' = 49: 0 (0.0)
|   |   |   |   |   |   'service' = 50: 1 (2.0)
|   |   |   |   |   |   'service' = 51: 0 (0.0)
|   |   |   |   |   |   'service' = 52: 0 (0.0)
```

```
|   |   |   |   |   |   'service' = 53: 0 (0.0)
|   |   |   |   |   |   'service' = 54: 0 (0.0)
|   |   |   |   |   |   'service' = 55: 0 (0.0)
|   |   |   |   |   |   'service' = 56: 0 (0.0)
|   |   |   |   |   |   'service' = 57: 1 (3.0)
|   |   |   |   |   |   'service' = 58: 0 (0.0)
|   |   |   |   |   |   'service' = 59: 0 (0.0)
|   |   |   |   |   |   'service' = 60: 0 (0.0)
|   |   |   |   |   |   'service' = 61: 1 (1.0)
|   |   |   |   |   |   'service' = 62: 0 (0.0)
|   |   |   |   |   |   'service' = 63: 0 (0.0)
|   |   |   |   |   |   'service' = 64: 0 (9.0)
|   |   |   |   |   |   'service' = 65: 0 (0.0)
|   |   |   |   |   |   'service' = 66: 0 (0.0)
|   |   |   |   |   |   'service' = 67: 0 (0.0)
|   |   |   |   |   |   'service' = 68: 0 (0.0)
|   |   |   |   |   |   'service' = 69: 0 (0.0)
|   |   |   |   |   |   'service' = 70: 0 (0.0)
|   |   |   'dst_host_serror_rate' > 0.76:
|   |   |   |   'count' > 1.0: 1 (193.0)
|   |   |   |   'count' <= 1.0:
|   |   |   |   |   'dst_host_same_srv_rate' <= 0.76: 1 (49.0/1.0)
|   |   |   |   |   'dst_host_same_srv_rate' > 0.76: 0 (7.0/1.0)
|   |   'src_bytes' > 0.0:
|   |   |   'dst_host_srv_serror_rate' > 0.0: 0 (38.0)
|   |   |   'dst_host_srv_serror_rate' <= 0.0:
|   |   |   |   'src_bytes' <= 8.0:
|   |   |   |   |   'dst_bytes' <= 103.0: 1 (1053.0/9.0)
|   |   |   |   |   'dst_bytes' > 103.0: 0 (46.0)
|   |   |   |   'src_bytes' > 8.0:
|   |   |   |   |   'src_bytes' <= 17.0: 0 (142.0/1.0)
|   |   |   |   |   'src_bytes' > 17.0:
|   |   |   |   |   |   'count' > 2.0: 1 (166.0)
|   |   |   |   |   |   'count' <= 2.0:
|   |   |   |   |   |   |   'service' = 1: 1 (0.0)
|   |   |   |   |   |   |   'service' = 2: 1 (0.0)
|   |   |   |   |   |   |   'service' = 3: 1 (0.0)
|   |   |   |   |   |   |   'service' = 4: 1 (0.0)
|   |   |   |   |   |   |   'service' = 5: 1 (0.0)
|   |   |   |   |   |   |   'service' = 6: 1 (0.0)
|   |   |   |   |   |   |   'service' = 7: 1 (0.0)
|   |   |   |   |   |   |   'service' = 8: 1 (0.0)
|   |   |   |   |   |   |   'service' = 9: 1 (0.0)
|   |   |   |   |   |   |   'service' = 10: 1 (0.0)
```

```
|   |   |   |   |   |   |   'service' = 11: 1 (0.0)
|   |   |   |   |   |   |   'service' = 12: 0 (6.0)
|   |   |   |   |   |   |   'service' = 13: 1 (0.0)
|   |   |   |   |   |   |   'service' = 14: 1 (0.0)
|   |   |   |   |   |   |   'service' = 15: 1 (116.0)
|   |   |   |   |   |   |   'service' = 16: 1 (5.0/1.0)
|   |   |   |   |   |   |   'service' = 17: 1 (0.0)
|   |   |   |   |   |   |   'service' = 18: 1 (0.0)
|   |   |   |   |   |   |   'service' = 19: 1 (0.0)
|   |   |   |   |   |   |   'service' = 20: 0 (17.0)
|   |   |   |   |   |   |   'service' = 21: 1 (0.0)
|   |   |   |   |   |   |   'service' = 22: 1 (0.0)
|   |   |   |   |   |   |   'service' = 23: 1 (0.0)
|   |   |   |   |   |   |   'service' = 24: 1 (0.0)
|   |   |   |   |   |   |   'service' = 25: 1 (0.0)
|   |   |   |   |   |   |   'service' = 26: 1 (0.0)
|   |   |   |   |   |   |   'service' = 27: 1 (0.0)
|   |   |   |   |   |   |   'service' = 28: 1 (0.0)
|   |   |   |   |   |   |   'service' = 29: 1 (0.0)
|   |   |   |   |   |   |   'service' = 30: 1 (0.0)
|   |   |   |   |   |   |   'service' = 31: 1 (0.0)
|   |   |   |   |   |   |   'service' = 32: 1 (0.0)
|   |   |   |   |   |   |   'service' = 33: 1 (0.0)
|   |   |   |   |   |   |   'service' = 34: 1 (0.0)
|   |   |   |   |   |   |   'service' = 35: 1 (0.0)
|   |   |   |   |   |   |   'service' = 36: 1 (0.0)
|   |   |   |   |   |   |   'service' = 37: 1 (0.0)
|   |   |   |   |   |   |   'service' = 38: 1 (0.0)
|   |   |   |   |   |   |   'service' = 39: 1 (0.0)
|   |   |   |   |   |   |   'service' = 40: 1 (0.0)
|   |   |   |   |   |   |   'service' = 41: 1 (0.0)
|   |   |   |   |   |   |   'service' = 42: 1 (0.0)
|   |   |   |   |   |   |   'service' = 43: 1 (0.0)
|   |   |   |   |   |   |   'service' = 44: 1 (0.0)
|   |   |   |   |   |   |   'service' = 45: 1 (0.0)
|   |   |   |   |   |   |   'service' = 46: 1 (0.0)
|   |   |   |   |   |   |   'service' = 47: 1 (0.0)
|   |   |   |   |   |   |   'service' = 48: 1 (0.0)
|   |   |   |   |   |   |   'service' = 49: 1 (0.0)
|   |   |   |   |   |   |   'service' = 50: 1 (5.0)
|   |   |   |   |   |   |   'service' = 51: 1 (0.0)
|   |   |   |   |   |   |   'service' = 52: 1 (0.0)
|   |   |   |   |   |   |   'service' = 53: 1 (0.0)
|   |   |   |   |   |   |   'service' = 54: 1 (0.0)
```

```
|   |   |   |   |   |   |   'service' = 55: 1 (0.0)
|   |   |   |   |   |   |   'service' = 56: 1 (0.0)
|   |   |   |   |   |   |   'service' = 57: 1 (0.0)
|   |   |   |   |   |   |   'service' = 58: 1 (0.0)
|   |   |   |   |   |   |   'service' = 59: 1 (0.0)
|   |   |   |   |   |   |   'service' = 60: 1 (0.0)
|   |   |   |   |   |   |   'service' = 61: 1 (0.0)
|   |   |   |   |   |   |   'service' = 62: 1 (0.0)
|   |   |   |   |   |   |   'service' = 63: 1 (0.0)
|   |   |   |   |   |   |   'service' = 64: 1 (0.0)
|   |   |   |   |   |   |   'service' = 65: 1 (0.0)
|   |   |   |   |   |   |   'service' = 66: 1 (0.0)
|   |   |   |   |   |   |   'service' = 67: 1 (0.0)
|   |   |   |   |   |   |   'service' = 68: 1 (0.0)
|   |   |   |   |   |   |   'service' = 69: 1 (0.0)
|   |   |   |   |   |   |   'service' = 70: 1 (0.0)
'src_bytes' > 28.0:
|   'src_bytes' <= 333.0:
|   |   'logged_in' = 1: 0 (6726.0/14.0)
|   |   'logged_in' = 0:
|   |   |   'src_bytes' <= 205.0:
|   |   |   |   'dst_bytes' <= 174.0: 0 (2737.0/11.0)
|   |   |   |   'dst_bytes' > 174.0:
|   |   |   |   |   'flag' = 1: 1 (0.0)
|   |   |   |   |   'flag' = 3: 1 (12.0/2.0)
|   |   |   |   |   'flag' = 4: 1 (0.0)
|   |   |   |   |   'flag' = 5: 0 (1.0)
|   |   |   |   |   'flag' = 6: 1 (0.0)
|   |   |   |   |   'flag' = 7: 1 (0.0)
|   |   |   |   |   'flag' = 8: 1 (0.0)
|   |   |   |   |   'flag' = 9: 1 (0.0)
|   |   |   |   |   'flag' = 10: 1 (0.0)
|   |   |   |   |   'flag' = 11: 1 (0.0)
|   |   |   |   |   'flag' = 2:
|   |   |   |   |   |   'same_srv_rate' <= 0.6: 1 (4.0)
|   |   |   |   |   |   'same_srv_rate' > 0.6: 0 (12.0/2.0)
|   |   |   'src_bytes' > 205.0:
|   |   |   |   'src_bytes' <= 230.0: 1 (43.0/2.0)
|   |   |   |   'src_bytes' > 230.0: 0 (20.0)
|   'src_bytes' > 333.0:
|   |   'src_bytes' <= 44788.0:
|   |   |   'logged_in' = 0:
|   |   |   |   'dst_bytes' > 1.0: 0 (88.0/2.0)
|   |   |   |   'dst_bytes' <= 1.0:
```

```
|   |   |   |   |   'service' = 1: 1 (0.0)
|   |   |   |   |   'service' = 2: 1 (0.0)
|   |   |   |   |   'service' = 3: 1 (0.0)
|   |   |   |   |   'service' = 4: 1 (0.0)
|   |   |   |   |   'service' = 5: 1 (0.0)
|   |   |   |   |   'service' = 6: 1 (0.0)
|   |   |   |   |   'service' = 7: 1 (0.0)
|   |   |   |   |   'service' = 8: 1 (0.0)
|   |   |   |   |   'service' = 9: 1 (0.0)
|   |   |   |   |   'service' = 10: 1 (0.0)
|   |   |   |   |   'service' = 11: 1 (0.0)
|   |   |   |   |   'service' = 12: 1 (0.0)
|   |   |   |   |   'service' = 13: 1 (0.0)
|   |   |   |   |   'service' = 14: 1 (0.0)
|   |   |   |   |   'service' = 15: 1 (0.0)
|   |   |   |   |   'service' = 16: 1 (565.0)
|   |   |   |   |   'service' = 17: 1 (0.0)
|   |   |   |   |   'service' = 18: 1 (0.0)
|   |   |   |   |   'service' = 19: 1 (0.0)
|   |   |   |   |   'service' = 20: 0 (205.0)
|   |   |   |   |   'service' = 21: 1 (0.0)
|   |   |   |   |   'service' = 22: 1 (0.0)
|   |   |   |   |   'service' = 23: 1 (0.0)
|   |   |   |   |   'service' = 24: 1 (0.0)
|   |   |   |   |   'service' = 25: 1 (0.0)
|   |   |   |   |   'service' = 26: 1 (0.0)
|   |   |   |   |   'service' = 27: 1 (0.0)
|   |   |   |   |   'service' = 28: 1 (0.0)
|   |   |   |   |   'service' = 29: 1 (0.0)
|   |   |   |   |   'service' = 30: 1 (0.0)
|   |   |   |   |   'service' = 31: 1 (0.0)
|   |   |   |   |   'service' = 32: 1 (0.0)
|   |   |   |   |   'service' = 33: 1 (0.0)
|   |   |   |   |   'service' = 34: 1 (0.0)
|   |   |   |   |   'service' = 35: 1 (0.0)
|   |   |   |   |   'service' = 36: 1 (0.0)
|   |   |   |   |   'service' = 37: 1 (0.0)
|   |   |   |   |   'service' = 38: 1 (0.0)
|   |   |   |   |   'service' = 39: 1 (0.0)
|   |   |   |   |   'service' = 40: 1 (0.0)
|   |   |   |   |   'service' = 41: 1 (0.0)
|   |   |   |   |   'service' = 42: 1 (0.0)
|   |   |   |   |   'service' = 43: 1 (0.0)
|   |   |   |   |   'service' = 44: 1 (0.0)
```

```
|   |   |   |   |   'service' = 45: 0 (8.0)
|   |   |   |   |   'service' = 46: 1 (0.0)
|   |   |   |   |   'service' = 47: 1 (0.0)
|   |   |   |   |   'service' = 48: 1 (0.0)
|   |   |   |   |   'service' = 49: 1 (0.0)
|   |   |   |   |   'service' = 50: 1 (0.0)
|   |   |   |   |   'service' = 51: 1 (0.0)
|   |   |   |   |   'service' = 52: 1 (0.0)
|   |   |   |   |   'service' = 53: 1 (0.0)
|   |   |   |   |   'service' = 54: 1 (0.0)
|   |   |   |   |   'service' = 55: 1 (0.0)
|   |   |   |   |   'service' = 56: 1 (0.0)
|   |   |   |   |   'service' = 57: 1 (0.0)
|   |   |   |   |   'service' = 58: 1 (0.0)
|   |   |   |   |   'service' = 59: 1 (0.0)
|   |   |   |   |   'service' = 60: 1 (0.0)
|   |   |   |   |   'service' = 61: 1 (0.0)
|   |   |   |   |   'service' = 62: 1 (0.0)
|   |   |   |   |   'service' = 63: 1 (2.0)
|   |   |   |   |   'service' = 64: 1 (0.0)
|   |   |   |   |   'service' = 65: 1 (0.0)
|   |   |   |   |   'service' = 66: 1 (0.0)
|   |   |   |   |   'service' = 67: 1 (0.0)
|   |   |   |   |   'service' = 68: 1 (0.0)
|   |   |   |   |   'service' = 69: 1 (0.0)
|   |   |   |   |   'service' = 70: 1 (0.0)
|   |   |   |   'logged_in' = 1:
|   |   |   |   |   'src_bytes' <= 334.0:
|   |   |   |   |   |   'count' > 2.0: 0 (31.0)
|   |   |   |   |   |   'count' <= 2.0:
|   |   |   |   |   |   |   'dst_bytes' <= 531.0: 1 (91.0)
|   |   |   |   |   |   |   'dst_bytes' > 531.0: 0 (5.0)
|   |   |   |   |   'src_bytes' > 334.0:
|   |   |   |   |   |   'dst_host_same_srv_rate' > 0.53: 0 (1677.0/22.0)
|   |   |   |   |   |   'dst_host_same_srv_rate' <= 0.53:
|   |   |   |   |   |   |   'dst_host_serror_rate' > 0.22: 0 (129.0)
|   |   |   |   |   |   |   'dst_host_serror_rate' <= 0.22:
|   |   |   |   |   |   |   |   'src_bytes' > 1298.0: 0 (443.0/6.0)
|   |   |   |   |   |   |   |   'src_bytes' <= 1298.0:
|   |   |   |   |   |   |   |   |   'src_bytes' <= 1194.0: 0 (372.0/12.0)
|   |   |   |   |   |   |   |   |   'src_bytes' > 1194.0: 1 (68.0/23.0)
|   |   |   'src_bytes' > 44788.0:
|   |   |   |   'src_bytes' <= 54540.0: 1 (190.0)
|   |   |   |   'src_bytes' > 54540.0:
```

```
|   |   |   |   'dst_bytes' > 164.0: 0 (19.0)
|   |   |   |   'dst_bytes' <= 164.0:
|   |   |   |   |   'src_bytes' <= 2280318.0: 0 (86.0)
|   |   |   |   |   'src_bytes' > 2280318.0: 1 (14.0)
```

C.2 Example prediction model

This example denotes a prediction model created by a classifier. A table is provided to further explain the output.

A simple prediction model for ten records will output as follows:

```
1,0
1,0
1,0
0,0
1,1
1,1
1,1
1,0
0,0
0,0
```

Table C.1 below further explains how a prediction model should be interpreted.

Table C.1: Example of a prediction model

Number	Ground Truth	Classifier Prediction
Prediction 1	1	0
Prediction 2	1	0
Prediction 3	1	0
Prediction 4	0	0
Prediction 5	1	1
Prediction 6	1	1
Prediction 7	1	1
Prediction 8	1	0
Prediction 9	0	0
Prediction 10	0	0

The prediction model is a binary classification task, as in this thesis, where 0 is normal traffic and 1 is malicious traffic. The *Ground Truth* column relates to the prediction within the Test data set and the *Classifier Prediction* relates to the classification by the machine learning algorithm.

APPENDIX D

DATA SET COMPARISON

Koch, Mario & Rodosek (2014) summarised all available data sets for intrusion detection research. The table has been adopted to include recent data sets and a popularity counter based on search results obtained from searching recent citations on Google Scholar.

Table D.1: Summary of intrusion detection data sets adopted from Koch et al. (2014)

Data set	Payload	Flows	Synthetic	Current Attacks	Ground Truth	Data set Availability	Citations Count	Reference
DARPA 98/99	Y	N	Y	N	Y	Y	62	Lippmann *et al.* (2000)
NSL-KDD	Y	N	Y	N	Y	Y	424	Tavallaee *et al.* (2009)
MAWI WORKING GROUP	Y	N	N	P	N	Y	52	Fontugne, Borgnat, Abry & Fukuda (2010)
CAIDA	Y	N	N	P	N	P	2	Team *et al.* (2009)
PREDICT/IMPACT	N	Y	Y	P	N	P	Not available	Not available
UNSW-NB15	Y	N	Y	Y	Y	Y	74	Moustafa & Slay (2015)

The values: Y denotes YES, P indicates partial and N indicates NO. A short literature review outlines that very limited labelled intrusion detection data sets are publicly available, as noted by Koch *et al.* (2014) and Małowidzki, Berezinski & Mazur (2015). The table was adopted to include the citation reference and recent citations for each data set. The results of the Google Scholar search was obtained by filtering the research citations for the data set research paper as outlined in figure D.1, Google Scholar citations. The search was performed on 24/09/2018 and filtered to select all citations since 01/01/2017. An additional inclusion for the UNSW-NB15 data set by Moustafa & Slay (2015) was made.

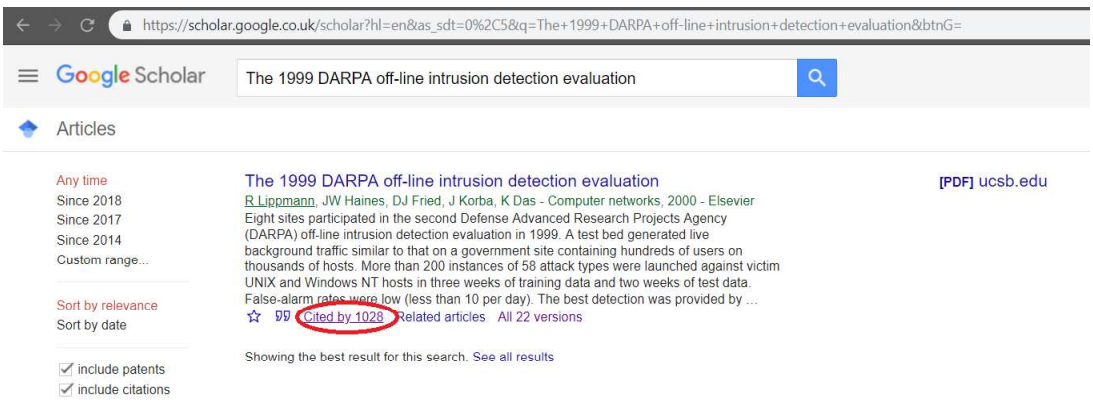

Figure D.1: Google Scholar advanced search

The ”Since 2017” parameter was changed for each data set’s results and the total recent citations extracted as per the below figure D.2, Google scholar filtered results. In the below instance, Darpa 98 data set was included in 62 research projects since 01/01/2017 and 24/09/2018.

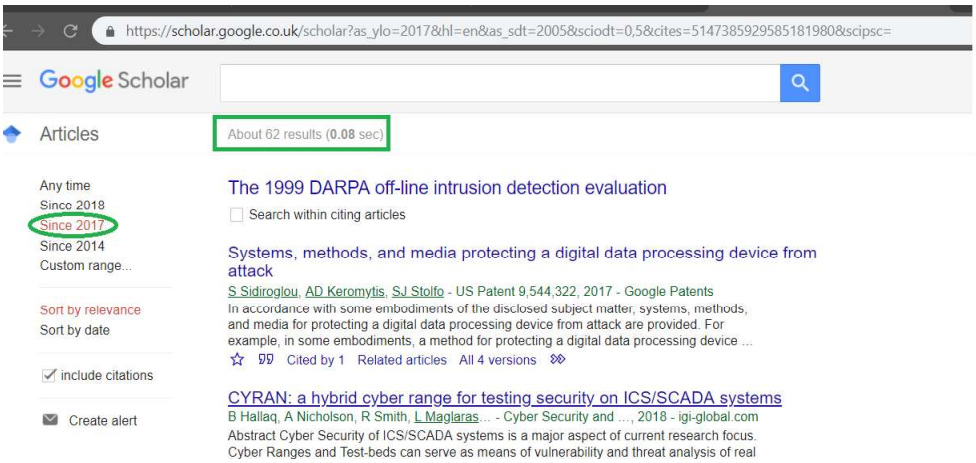

Figure D.2: Google Scholar filtered results