DETECTION AND CLASSIFICATION OF MALICIOUS NETWORK STREAMS IN HONEYNETS

A thesis presented in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
at Massey University, Palmerston North,
New Zealand.

Fahim U.H. Abbasi

2013

Abstract

Variants of malware and exploits are emerging on the global canvas at an ever-increasing rate. There is a need to automate their detection by observing their malicious footprints over network streams. Misuse-based intrusion detection systems alone cannot cope with the dynamic nature of the security threats faced today by organizations globally, nor can anomaly-based systems and models that rely solely on packet header information, without considering the payload or content.

In this thesis we approach intrusion detection as a classification problem and describe a system using exemplar-based learning to correctly classify known classes of malware and their variants, using supervised learning techniques, and detect novel or unknown classes using unsupervised learning techniques. This is facilitated by an exemplar selection algorithm that selects most suitable exemplars and their thresholds for any given class and a novelty detection algorithm and classification algorithm that is capable to detect, learn and classify unknown malicious streams into their respective novel classes. The similarity between malicious network streams is determined by a proposed technique that uses string and information-theoretic metrics to evaluate the relative similarity or level of maliciousness between different categories of malicious network streams. This is measured by quantifying sections of analogous information or entropy between incoming network streams and reference malicious samples. Honeynets are deployed to capture these malicious streams and create labelled datasets. Clustering and classification methods are used to cluster similar groups of streams from the datasets. This technique is then evaluated using a large dataset and the correctness of the classifier is verified by using "area under the receiver operating characteristic curves" (ROC AUC) measures across various string metric-based classifiers. Different clustering algorithms are also compared and evaluated on a large dataset.

The outcomes of this research can be applied to aid existing intrusion detection systems (IDS) to detect and classify known and unknown malicious network streams by utilizing information-theoretic and machine learning based approaches.

Acknowledgements

All praises and thanks to Allah the exalted, the creator and lord of the universe, the source of all knowledge and wisdom. I thank Allah for all His blessings and especially for blessing me with good health, audacity, potential and ability to complete this manuscript.

I wish to express my profound gratitude to my supervisors Prof. Richard Harris, Dr. Giovanni Moretti and Prof. Stephen Marsland for their time, wisdom, patience, keen interest, inspiring guidance and valuable suggestions in planning, conducting and writing this manuscript. Thank you for sharing your wealth of knowledge with me and thank you for teaching me scientific methods and research. I would like to thank all my teachers who taught me in school, college and university in Pakistan and New Zealand. You all played your part that helped me to achieve my goals.

I have no words to express my gratitude to my father Mansoor-ul-Huda Abbasi and my mother Dr. Arsala Mansoor for all they have done for me. Thank you for believing in me and always encouraging me to pursue a Phd. I wish my father was alive and could be with us today to cherish this achievement. Both my parents were awarded a prestigious scholarship for Postgraduate studies overseas (in 1985) and I am glad to be their flag bearer. I would like to thank my siblings, my sister Asra, brother-in-law Irfan, and my brother Ibrahim. Thank you for your wishes and support.

I would like to thank my wife Rooshan, who has always stood firm by my side through good and difficult times during my PhD. Your constant love and support played a major role in completion of my PhD.

I would like to thank my lovely daughters Fatima and Sara. You have provided the perfect distraction that allowed me to keep my sanity and provided a good motivation to finish my thesis.

I would like to thank my fellow Postgrads at Massey University, especially my office mates Graham Jenson and Jevon Wright for the valuable discussions and distractions.

I would like to thank my fellow Pakistani students and the Pakistani Palmerston North community for all their support and help, especially Yousuf, Jawad, Shujjat, Aamir, Jibran, Ibrar and Dr. Zulfiqar Butt. Thank you for all the support, festivities and distractions that made us feel at home.

I would like to thank the Higher Education Commission (HEC) of Pakistan for funding my studies at Massey University through the MS leading to PhD scholarship. You provided a very strong motivation to persue a PhD at Massey University NZ.

I would like to thank Massey University and the School of Engineering and Advanced Technology (SEAT) for giving me the opportunity to do a PhD degree.

I would like to the thank the SEAT staff especially Michele Wagner, Linda Lowe, Dilantha Punchihewa and Gayle Leader for sorting out numerous administrative and financial issues during the tenure of my PhD. and especially the SEAT support staff: Patrick Rynhart, Colin Plaw, Mike Lusby and Nick Look. Your technical expertise and discussions have helped me during the thick of things.

Finally I would like to thank Dr Ian Welch, Dr Peter Komisarczuk, Dr Felix Leder and Dr. Tillmann Werner for their support and advice.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"The wire protocol guys don't worry about security because that's really a network protocol problem. The network protocol guys don't worry about it because, really, it's an application problem. The application guys don't worry about it because, after all, they can just use the IP address and trust the network."

– *Marcus J. Ranum*

This chapter presents an introduction and background to the information security problem followed by the motivation for this thesis. The challenges in intrusion detection schemes are identified and the need for intrusion detection and classification of malicious network streams is described. The scope of the research is presented and research problems are highlighted. The research goals are identified followed by an account of the contributions and limitations of this thesis. The chapter concludes with an outline of this thesis.

1.1 Background

In the past, information security has continually been overlooked and not considered as a real problem. The need to secure data and information assets at a business and consumer level was drawn into public consciousness after the commercialization of the Internet in the late 1980s. Innella (2000) correlated this with emergence of the first virus in 1988, and states:

"Therefore, in the fall of 1988 the world saw evidence of the true threats that existed to

network security. The Internet Virus[1] *was launched at that time and all of the 60,000 computers on the Internet were crippled for two entire days."*

Before that, there were no real threats, as the early Internet was shared between very few universities and government organizations primarily to conduct, collaborate and share research (Innella, 2000).

Security is not an out-of-the-box solution. It requires careful analysis of the environment at hand before a solution can be proposed. It is a layered process and demands a thorough understanding of the system and its constraints. No system is 100 percent secure, the security of a system is as strong as its weakest point (Marcinkowski, 2001). Denning (1987) modelled the security problem around information and partitioned it into three major categories, more commonly known as the "CIA triad" or "Confidentiality", "Integrity" and "Availability":

1. Confidentiality: Private sensitive information designated for certain users is available only to those authorized users in an authorized manner. This may include file permissions, access control lists, encryption and other means.

2. Integrity: Information from trusted sources can be trusted. This means that the information is protected from deletion or modification by unauthorized users. Both Confidentiality and Integrity of a system can be attacked by unauthorized access to private information and services.

3. Availability: Information is accessible and available to authorized users when needed. This ensures that information resources are available to serve information to authorized users whenever they require that information. Attacks such as Denial Of Service (DOS)[2] attacks may result in service outages causing delays to access information.

Attackers take advantage of networks and use them as a mass transit system to exploit remotely accessible services and spread their automated malware globally, in the form of "malicious streams"[3]. Intrusion detection systems (IDSs) have become an essential tool in the arsenal of security professionals responsible for protecting production networks. IDSs have the capability to detect these malicious streams flowing through public and private networks.

[1] "Self-replicating program that propagates through some form of infected files" (Hansman and Hunt, 2005)

[2] "An attack which prevents legitimate users from accessing or using a host or network." (Hansman and Hunt, 2005)

[3] Network streams created with malicious intent that may contain attacks, exploits, shellcode, or malware

Intrusion detection is the art of detecting malicious activity in a computer-related system (Phoha, 2002). Malicious activities and intrusion techniques are interesting from a computer security perspective. Denning (1987) classified IDS into "host-based" and "network-based" IDS. Although recent progress and research in anomaly-based IDS is quite remarkable, the majority of networks today still rely heavily on misuse-based IDS for their network security.

Anomaly detection is the art of finding patterns in data that do not conform to expected behaviour or models (Chandola et al., 2009). Analysis of traffic and events reveals that intrusion behaviour patterns are different from the normal behaviour of system usage, and hence anomaly detection techniques are applicable to intrusion detection. The approach is to build models of normal data and detect deviations in observed data.

Misuse or signature-based detection relies on pre-defined content or pattern matching techniques, in the form of signatures of known attacks. Signature matching techniques are used to identify attacks by comparing the contents of packets with a set of signatures or rules. These signatures comprise of distinct syntactic features observed from attacks as they occur or have occurred. These features are in the form of patterns or sub-strings. These patterns comprise of symbols or information, which make them unique from other signatures and are used to "describe the characteristic elements of an attack" (Kreibich and Crowcroft, 2004).

Signatures are created after analysing attack traffic data. Signature creation process is labour intensive and requires careful analysis of the malware or exploit. A signature is considered effective if it can narrow down the attack characteristics while being elastic enough to detect some variations in the attack (Kreibich and Crowcroft, 2004). A good signature should be specific enough to narrow down the exploit and avoid miss-detection leading to false negatives and yet be generic enough to catch variants of the same exploit, while avoiding classification of benign traffic as malicious, leading to false positives.

1.2 Motivation

The unprecedented growth in the use of Information Technology (IT) has resulted in a multitude of digital assets in the form of applications and services. Business enterprises have built their entire business models utilizing these digital assets. Networks have evolved from low speed half-duplex links to full-duplex, multi-homed, self-convergent, gigabyte streams, controlled by advanced protocols. Applications and services have also evolved from static, low-featured, with limited usability applications to highly interactive, advance, dynamic, multi-featured, high-usability, extendible and intelligent applications. The security of these applications and services, accessible over interconnected networks, poses a major challenge to the IT industry. Our IT infrastructure and resources are threatened on a daily basis by worms[4], viruses, botnets[5] and even state-sponsored cyber-militants.

Each day, attackers or crackers[6] and their automated agents such as: worms, viruses, trojans[7] and bots more commonly known as "malware" or malicious software, exploit numerous vulnerabilities, threatening the IT infrastructure and its associated digital assets. The attacker's motivation can range from pranks, to personal gain, to lucrative business models and cyber-espionage. The goals, however, are the same: to gain unprivileged access to private data, information and computing resources. The attackers have evolved from teenage "script kiddies" (Mead and Stehney, 2005), motivated to gain publicity and notoriety by spreading havoc, to organized gangs driven by financial motives, competing in a highly lucrative underground industry with a flourishing economy. Sensitive user data is the raw material for this industry and is continually being harvested both overtly and covertly from globally distributed computers at an industrial scale. Attackers have also evolved into state sponsored cyber-militia, involved in sophisticated cyber-warfare, targeting the critical infrastructure of other nation states.

Over the years, we have observed a sharp increase in the intricacy, sophistication and overall frequency of attacks. The availability of user friendly hack tools has increased the number of attacks which do not demand a thorough understanding from their users. Lipson (2002) studied and correlated this trend graphically, an extended version of which is illustrated in Figure 1.1.

The Internet community has witnessed a monumental increase in cyber security incidents over the last 2 decades. This trend is illustrated in Figure 1.2 from the number of incidents reported as statistics from the Computer Emergency Response Team Coordination Center (CERT/CC).

Symantec, a well-known anti-virus company, published its 2011 Internet Security Threat Report Symantec (2011). A few highlights from this report are mentioned here:

- Blocked over 5.5 billion malware[8] attacks in 2011.

[4] "Self-replicating program that propagates through network services on computers or through email". (Hansman and Hunt, 2005)

[5] "A group of computers that have been compromised and brought under the control of an individual." `http://us.norton.com/security-glossary/article`

[6] "Someone who tries to break the security of, and gain access to, someone else's system without being invited to do so." (Shirey, 2000)

[7] "A program made to appear benign that serves some malicious purpose". (Hansman and Hunt, 2005)

[8] "A contraction of "malicious software"". (Shirey, 2000)

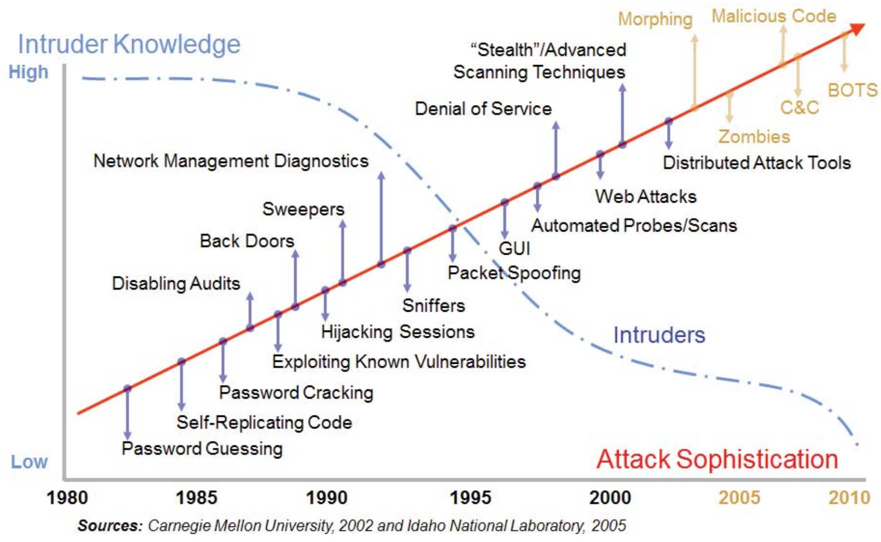


Figure 1.1: Intruder knowledge vs Attack sophistication (Lipson, 2002)

- 403 million of these attacks were caused by new variants of malware.
- Approximately 4500 web attacks per day.
- Nearly 42 billion spam messages.
- Over 5000 new vulnerabilities.
- Over 232.4 million identities were exposed overall during 2011.
- Cost due to the loss of business and information caused by these security breaches accrued to billions of dollars globally.

A few notable information security incidents of global importance are presented here:

- It has been reported[9] that since 1995 to 2007, the Peoples Liberation Army of China has been involved in cyber-espionage involving stealing secret information and theft of intellectual property from public and private institutes in the US .
- Operation Shady RAT [10] initiated in 2006, in which over 70 government and corporate businesses across 14 countries were attacked and their digital assets controlled by a single actor, allegedly China.
- In 2007, Estonia's cyber infrastructure came under cyber attacks originating from Russia. [11]

[9]Report to Congress:`http://www.defense.gov/pubs/pdfs/2010_CMPR_Final.pdf`
[10]`http://www.mcafee.com/us/resources/white-papers/wp-operation-shady-rat.pdf`
[11]`http://www.guardian.co.uk/world/2007/may/17/topstories3.russia`

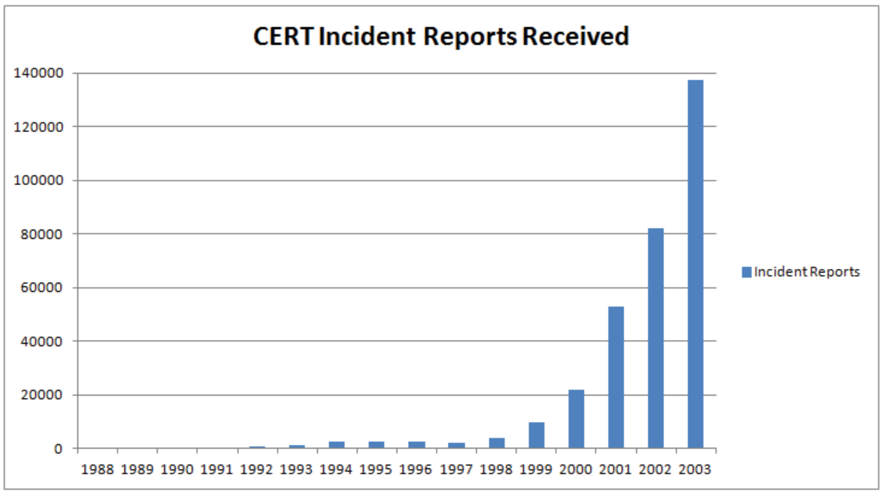


Figure 1.2: CERT Incident reports received from 1988-2003 (Hansman and Hunt, 2005)

- In 2010, hackers from Pakistan defaced 1000+ Indian government and business websites in response to similar attacks conducted earlier by the Indian Cyber Army. [12]
- In 2010, Iran's Natanz nuclear enrichment facility was attacked by the Stuxnet worm [13], which damaged centrifuges used in the nuclear plant. It is believed that this worm was designed jointly by the Americans and the Israelis as an attempt to undermine Irans nuclear program.
- In 2010, Operation Aurora [14] allegedly operated by China, targeted many US defence and other companies including Google, Adobe, Juniper, Yahoo, Symantec and others. The target was to steal intellectual property in the form of source code from these companies.
- In 2012, the Maher Center in Iran reported a complex and highly sophisticated cyber-attack in the form of "Flame" [15] spyware, that was infiltrating Iranian computers and stealing and sending data to a remote command and control system.

Response time is critical in the advent of an intrusion. While available security tools provide a good set of static defences, they cannot cope with the dynamic nature of the threats, nor can they propose dynamic responses. Malware is getting more sophisticated, harder to detect and remove. New attacks and vulnerabilities are emerging on

[12]http://tribune.com.pk/story/83967/36-government-websites-hacked-by-indian-cyber-army/
[13]http://www.nytimes.com/2012/06/01/world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html
[14]http://www.mcafee.com/us/threat-center/operation-aurora.aspx
[15]Flame:http://edition.cnn.com/2012/05/29/tech/web/iran-spyware-flame/index.html

a large scale. AV companies generate over 100,000 signatures a week [16]. Attackers use attack variants to easily evade detection (Ptacek, 1998). Evasion can be achieved by small changes in the attack patterns that undermine detection mechanisms based on byte-by-byte matching. In order to identify these small changes, security analysts need to carefully study the attacks which lead to security incidents. This may require setting up of an environment in which network and system level events and details are logged extensively. Security tools such as: Anti-virus software, firewalls, and intrusion detection systems (IDSs) are important tools to combat these attacks, but are greatly limited due to a vast number of false positive (caused by miss-classification) and false negative (caused by non-detection) results and thus give a false sense of security. The work in this thesis focuses on the problems of network monitoring tools such as Network-based Intrusion Detection Systems (NIDS) and proposes a solution based on payload-based analysis techniques. These problems motivated me to investigate:

1. Resources and methodologies to setup environments that can invite attackers and collect real attack traffic and data.
2. Methods which are more flexible, adaptable and resilient to cope with minor changes in the network payload to detect attack variants with confidence. This lead to the investigation of information-theoretic measures and string metrics.
3. Methodologies to classify known attacks and their variants from network streams, using supervised learning techniques.
4. Methodologies to detect and classify new and novel attacks and recommend them to the security analyst for analysis. This involved investigating unsupervised learning methods.

1.3 Network-based Intrusion Detection Research Challenges

1. In order to detect an intrusion, a misuse-based IDS relies on signatures it has in its signature database to compare with incoming network streams and packets. Signature generation is a laborious process. It may require hours of analysis of attack traffic by a human analyst, until a final effective signature can be produced and pushed out to be added to the database used by the IDS (Phoha, 2002). Meanwhile the automated attacks in the form of an exploit or malware is continuously spreading havoc by globally infecting computer systems.

[16]http://www.zdnet.com/blog/bott/the-malware-numbers-game-how-many-viruses-are-out-there/4783

2. Most intrusion detection methods tend to be static, rule-based systems. They typically match against the signatures of known threats in packet content and apply rules to drop, forward, log, or accept such packets. Signature-based detection alone cannot cope with the demands of the current threat landscape. Attackers today are quite capable of undermining anti-malware software and thwarting intrusion detection systems (Ptacek, 1998). Scrambling information within packets can effectively evade misuse based IDS. Slight variations in the expected patterns can result in the intrusion evading detection (Ptacek, 1998). The most common methods of introducing variation into packet data are data scrambling or transposition, where the order of symbols forming the pattern changes, but the inherent symbols remain the same. These syntactically distinct, but symbolically similar, strings have similar entropy measures and so can be identified in this way using information theoretic measures such as the Kolmogorov complexity (Cilibrasi and Vitányi, 2005). However, this does not work if new symbols are added, or the old ones are re-encoded, although other string metrics could be useful here.

   Sometimes, even small variations result in new attacks which do not get detected by the IDS. A similar situation occurs for new unknown zero-day exploits and thus require a new iteration of incident or attack analysis for signature generation. Ideally, the process to learn signatures should be automated to allow an enterprise to stop an imminent attack. This requires that a system should automatically perform intelligent traffic payload analysis to identify unique characteristics that can help in generating signatures for IDS, or recommend suspicious or malicious traffic to the human expert. This may involve identifying known attacks and learning unknown ones.

   The derivation of a sufficiently specific signature is not easy. However, simply looking for signature matches is also problematic, as minor changes in the packet data can lead to the data being sufficiently different such that it is no longer identified by the signatures. This limitation is actively exploited by malware writers (Ptacek, 1998). However the signatures cannot be made too generic as this leads to false positives, which greatly undermines confidence in the IDS and can result in it being ignored or disabled.

3. Existing anomaly-based solutions have a similar problem, with the addition that most do not take into account any payload contents. Most anomaly detection systems rely on statistical features extracted from header infromation such as: Source and destination IP and Port, traffic volumes, arrival rates, IP and Port distribution and latency. Packet header checks alone cannot promise security, without incorporating entire packet payloads. Security policies based only on

header information such as: IP address, Ports, and Protocols may become unreliable as attackers may conceal their malicious operations using ports used by legitimate services, and use that trust model to transport malware via the payload. Thus, obfuscating their malicious operations. IDS now need to be adaptable and anomaly, payload or content aware.

4. Stateful protocol analysis techniques involve matching of each connection with an existing template that acts as a profile for a given protocol such as FTP[17], HTTP[18], SMTP[19] and others. Any deviations from this profile are immediately reported. A limitation is that attacks conforming to normal protocol behaviour tend to go unnoticed (Rehak et al., 2009), and this can be used by attackers to their advantage.

1.4 Scope of study

This thesis shows the effectiveness of information-based intrusion detection techniques that involve string matching and compression on Honeypot network data. The proposed system is designed to aid existing signature-based intrusion detection systems, with the additional capability to detect variations of network intrusions in the form of malicious streams of trojans, viruses, worms, shellcode and other active threats. To enable intelligent learning for automated classification, exemplar-based learning techniques are investigated to detect variants of known attacks while they are being transferred over a network. Similarity between known malicious stream samples and the incoming traffic is calculated based on similar information or similar symbols. Such synergies are thus quantified to yield a similarity score with a certain level of confidence and this is used to classify examples of known attacks, and to add new classes of attack where there is no known exemplar, so that the system can engage in lifelong learning and continue to extend itself. The system is integrated with our existing Honeynet setup and we demonstrate our approach using data from real Honeypots. In this thesis, the research scope is restricted to intrusion detection schemes that can perform packet payload analysis. This involves:

- Collection of malicious network streams for ground truth.
- Detection of malicious network streams and their variants (Supervised Learning).
- Detection of new or novel malicious network streams (Unsupervised Learning).

[17]File Transfer Protocol
[18]Hypter-text Transfer Protocol
[19]Simple Mail Transfer Protocol

- Classification of all these streams.

1.4.1 Collection of malicious network streams for ground truth

In order to evaluate an IDS, it is necessary to have or create a dataset to serve as a ground truth. This dataset should comprise of known malicious streams and their variants. Variants can be synthesized or captured from live traffic. Details of known malicious streams are studied and collected from sources such as:

Existing documented sources

It is important to start with known samples of malicious streams that can be labelled and verified to train and test our system. Details of known malicious streams are studied and collected from existing knowledge sources such as: CVE[20], CERT[21], Snort Signatures[22] and existing datasets collected by researchers and organizations (discussed in Chapter 2). These knowledge sources have detailed documentation of the attack or exploit, but often do not have live attack samples. These sources can be used to craft or synthesize attack traffic and their variants.

Live sensors

Most network security tools are passive in nature; for example, firewalls and IDS. They operate on available rules and signatures in their database. Anomaly detection is limited only to these sets of available rules. Any activity not in alignment with those rules goes undetected. In order to achieve a better insight into the attacks and attacker tactics, there is a need to set up a vulnerable environment that lures an attacker, to study their behaviour. To this end, apparently benign computer systems designed to keep detailed logs of system activity are widely deployed today by security researchers. These systems, known as 'Honeypots', are designed to record a hacker's activities to gain an insight into the methods used. The logs typically would include intruder keystrokes, processes, and system-wide and network data. Over the years, researchers have successfully isolated and identified worms and exploits using Honeypots placed in specialized architectures called Honeynets. Honeynets are capable of logging information, far more effectively than any other available security tool, providing the

[20]Common Vulnerabilities and Exposures: `http://cve.mitre.org/`
[21]Computer Emergency Response Team: `http://www.us-cert.gov/ncas/alerts/`
[22]Snort Rules: `http://www.snort.org/snort-rules/`

capability to study hackers “under a microscope”. We have deployed multiple Honeypot sensors emulating various services such as: FTP, HTTP, SSH, DNS[23], SMB[24], RIP[25], DHCP[26], NTP[27], POP[28], TELNET, SOCKS, SNMP[29], IRC[30], RPC[31], RDP[32], VNC[33], XMPP[34] and others to collect malicious network streams.

These live sensors provide three main advantages over other techniques:

- They provide an advantageous position for attack analysis, as the researchers can control both the endpoints (Honeypots) and the intermediary network (Honeynet).
- Since they have no production value, any connection attempt to them is considered suspicious or malicious. Thus avoiding intensive pre-filtering, which may be required in production networks.
- They serve as an excellent source to collect live samples of malicious network streams and malware.

1.4.2 Detection of malicious network streams and their variants: Supervised Learning

Once a ground truth, in the form of a labelled dataset of malicious network streams and their variants has been created, the next challenge is to investigate methods to detect similarity between known malicious streams and their variants. The key research questions addressed are:

Q: Is it possible to measure similarity between samples of malicious network streams?

Q: Is it possible to measure similarity between variants of malicious network streams with statistical confidence?

Q: How should similar samples be grouped?

[23]Domain Name Service
[24]Server Message Block
[25]Routing Information Protocol
[26]Dynamic Host Configuration Protocol
[27]Network Time Protocol
[28]Post Office Protocol
[29]Simple Network Management Protocol
[30]Internet Relay Chat
[31]Remote Procedure Call
[32]Remote Desktop Protocol
[33]Virtual Network Computing
[34]Extensible Messaging and Presence Protocol

Q: How should appropriate exemplars from known malicious stream samples for classification be selected: Instance selection problem?

1.4.2.1 Similarity measurement between malicious network streams: The use of information-theoretic and string metrics based measures

One of the key requirements for classification is to measure similarity between malicious network streams. String metrics, information-theoretic measures and other methods are investigated, that can be applied on the payload collected from malicious network streams for similarity measurements. The key questions addressed include:

Q: How should similarity between similar types of malicious network traffic or streams be measured?

Q: What metrics are available?

Q: How do these metrics compare?

Q: What metrics provide the most appropriate methods for making similarity comparisons?

1.4.3 Detection of new or novel malicious network streams: Unsupervised Learning

Techniques have been studied and adopted that involve detection and handling of new or novel streams which may appear as a result of classification. The key questions addressed here include:

Q: How should unknown or novel samples that did not get classified by prior knowledge extracted from the training set (TR) be identified?

Q: How should appropriate exemplars from these unknown or novel samples for classification be selected?

Q: How should groups and sub-groups that may exist in these novel samples be identified to carry out classification?

1.4.4 Stream Classification

Clustering and classification algorithms have been studied, proposed and evaluated to classify same, similar and novel streams from any given dataset. Evaluation for the accuracy of these algorithms is conducted using ROC curve analysis. Best performing algorithms, that exhibit the best results in terms of highest true positive rate (TPR) and lowest false positive rate (FPR) are considered.

1.5 Research Goals

The aim of this research is to develop novel information-theoretic intrusion detection techniques for the detection and classification of:

- Known malicious network streams. (Supervised Learning discussed in Chapters 4 and 5)
- Variants of known malicious network streams. (Supervised Learning discussed in Chapters 4 and 5)
- New or novel malicious network streams. (Unsupervised Learning discussed in Chapter 6)

The proposed system is designed to aid existing signature-based intrusion detection systems, with the additional capability to detect variations of network intrusions and novel intrusions.

Samples of malicious streams are collected using Honeynets and labelled. Techniques for determining similarity between malicious streams are investigated. Clustering and classification algorithms are used for clustering and classification of similar malicious streams. Supervised learning approaches are used for classification of known malicious streams and their variants, while unsupervised learning approaches are used for the classification of new or novel malicious or suspicious streams.

1.6 Contribution of this thesis

1. Developed a means to contain and capture attacker activity in a controlled environment.
2. Developed a method to measure the similarity between malicious and benign network streams with statistical confidence.
3. Developed the capability to detect variants of known malicious streams with confidence and classify them.

4. Developed the capability to detect new and novel streams.
5. Developed the capability to classify these novel streams into groups and subgroups.

1.7 Limitations

The system proposed in this thesis has some limitations, which are listed here:

1. Currently the system is not capable of detecting encrypted streams, encrypted using substitution ciphers.
2. It might not reliably detect compressed traffic. May lead to partial or no detection.
3. Not tested on binary malware.
4. Not capable of automatically assign a meaningful label to novel streams. Needs supervisory guidance.
5. No means to automatically validate the ground truth.
6. No mechanism to automatically validate malicious exemplars.
7. The system is not real-time.

These limitations could be eliminated or reduced by further exploration and investigation possibly following thoughts, directions and suggestions outlined in the future work section.

1.8 Thesis Structure and Outline

The rest of this thesis is structured as follows:

Chapter 2. This chapter presents a theoretical background and gives an account of the state of the art in intrusion detection technologies. This is followed by a brief history and detailed taxonomy, in which we attempt to position our research context. Computer attacks are studied to label and categorize attacks to form a ground truth. Techniques to collect datasets or ground truth for IDS benchmarking are discussed and Honeypot and Honeynet technologies are introduced as tools to collect real and live attack data.

Chapter 3. This chapter presents the design and architecture of our proposed IDS. Discusses evaluation techniques for IDS and presents the dataset that we collected and use. This is followed by a discussion on the usefulness of Honeynet and Honeypot technologies, their design challenges, and a discussion on our implementation followed by results in the form of intrusion data collected from this setup.

Chapter 4. This chapter explores what similarity metrics are available for similarity measurement of malicious network streams, discusses their advantages and disadvantages and provides experimental evidence to select the best metrics.

Chapter 5. This chapter discusses the advantages and disadvantages of various clustering and classification algorithms that can be used for intrusion detection. It proposes a clustering scheme fit for our domain and evaluates, compares and tests these schemes to find the best one, based on an appropriate criterion.

Chapter 6. This chapter deals with machine learning techniques that can be used to add automated learning to our intrusion detection system. Exemplar based learning techniques are investigated and custom algorithms are created to deal with both supervised learning and unsupervised learning scenarios. Novelty detection strategies are also discussed and implemented. The chapter concludes with a comparison of the proposed algorithms with other known techniques for evaluation and verification.

Chapter 7. This chapter summarizes the research contribution and gives a conclusion to the research presented in this thesis and points out directions for future work and improvements.

Chapter 2

Literature Review on Intrusion Detection

"Know thy self, know thy enemy. A thousand battles, a thousand victories."
–Sun Tzu, The Art of War

This chapter reviews the state of the art in intrusion detection technologies, their history, types and taxonomy and their strengths and weaknesses. It introduces computer attacks and discusses the need to study them for classification and labelling purposes. Data collection methodologies for benchmarking intrusion detection systems are studied and Honeypot and Honeynets technologies are presented as useful tools for data collection and attack analysis.

2.1 Intrusion Detection Systems

The Internet Security Glossary (RFC2828) defines an "Intrusion" as:

"A security event, or a combination of multiple security events, that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system (or system resource) without having authorization to do so." (Shirey, 2000)

The same document defines an "Intrusion Detection System (IDS)" as:

"A security service that monitors and analyses system events for the purpose of finding, and providing real-time or near real-time warning of, attempts to access system resources in an unauthorized manner." (Shirey, 2000)

Intrusions occur as attackers and intruders[1] use sophisticated means to exploit vulnerable systems to:

- Gain unauthorized remote access to computer hosts over the network or Internet.
- Gain unauthorized additional user privileges on a host system.
- Misuse privileges granted for malicious gain.

IDS enable organizations and enterprises to protect their system from the onslaught of attackers. On detecting an attack or intrusive behaviour, an IDS can be configured to respond, either by logging details of the attack and the attacker, or by actively taking automated actions to mitigate further spread of the attack. Careful consideration needs to be taken while configuring an IDS. Bad or misconfiguration of an IDS may lead to either too many alarms or False positives (FP), or undetected attacks or False Negatives (FN). Such situations are detrimental for the security of the organization.

Intrusion detection is not a plug-and-play or an out-of-the-box solution. Judy Novak highlights this as: "Intrusion detection is not a specific tool but a capability, a blending of tools and techniques" (Novak and Northcutt, 2003). It requires expert level knowledge, skill and experience to install, configure and maintain such systems in a home or enterprise environment.

2.1.1 A Brief History and Review Of IDS

The literature suggests over time there has been a transition from traditional host-based statistical and pattern matching intrusion detection schemes to network-based intrusion detection sensors to hybrid data mining and machine learning approaches. Techniques involving information-theoretic measures have shown promise and have recently emerged as simpler techniques for payload or content-based intrusion detection. For NIDS[2], it has been observed that most techniques rely on features extracted from packet header data. Such techniques have become unreliable and easy to evade (Ptacek, 1998). Researchers are now employing techniques that involve payload analysis in order to ensure that evasion is more difficult.

Early Host-based IDS (HIDS): In 1980 Anderson (1980), while working for the US government suggested the notion of an automated program for computer security

[1] "An entity that gains or attempts to gain access to a system or system resource without having authorization to do so" (Shirey, 2000)

[2] Network Intrusion Detection System

threat monitoring and surveillance. His focus was on an automated review of UNIX system or host-based audit trails, to optimize them and reduce redundancies within them. Although he did not describe his system as an Intrusion Detection System, but functionally laid down the foundation of such a system by pointing out the need for statistical analysis of user behaviour. The development of IDS initiated with a focus on host-based IDS. Dorothy Denning (1987) described the first model and framework of an Intrusion Detection System (IDS) in her seminal paper in 1987, while she was working at SRI International. This led to the development of a host-based IDS called "Intrusion Detection Expert System (IDES)" (Javitz and Valdes, 1991) at SRI, this was a hybrid system that relied on anomaly detectors and expert systems. In 1988, the Lawrence Livermore Labs at UC Davis developed the "Haystack" IDS (Smaha, 1988) for the US Air Force. The same year the National Computer Security Center (NCSC) developed another IDS called "Multics intrusion detection and alerting system (MIDAS)" (Sebring et al., 1988). In 1989, Los Alamos National Laboratory developed a statistics-based anomaly detector called "Wisdom and Sense (W&S)". IDES, MIDAS, Haystack and W&S were host-based IDS, that relied on user behaviour and system log information analysis for detection. These initial IDS were effective systems in their time, but cannot meet the requirements of the current threat landscape and sophisticated malware that exists today. This is because these systems relied on host-based detection alone and current day malware is intelligent enough to employ techniques to imitate normal user behaviour while evading detection.

Network-based IDS (NIDS): During the 1990's, Intrusion Detection Systems started to mature, demonstrating a move towards more sophisticated, network-centric and hybrid approaches. In 1990, Heberlein at UC Davis developed the first network-based IDS called "Network Security Monitor (NSM)" (Heberlein et al., 1990). This system laid the foundation for all network-based intrusion detection systems, as it involved packet capture and analysis. In 1991, UC Davis developed the first hybrid IDS called "Distributed Intrusion Detection System (DIDS)" (Snapp et al., 1991). DIDS used NSM for network-based detection and Haystack for multi host-based detection. The same year Los Alamos National Laboratory developed the "Network Anomaly Detection and Intrusion Reporter (NADIR)" (Ning and Xu, 2003), a network-based IDS. NADIR performs detection by analysing audit trails of network service nodes. In 1993, SRI extended IDES to create an intelligent host-based IDS called "Next-generation Intrusion Detection Expert System (NIDES)". UC Santa Barbara developed the STAT and USTAT (Ilgun, 1993) system, a host-based IDS with a focus on transition analysis based detection in 1993. Researchers at Purdu University applied state-transition-based analysis with coloured petri-nets to IDS to create "IDIOT" (Crosbie et al., 1996). In 1996

DARPA sponsored the "Graph-based IDS (Gr-IDS)" project at UC Davis to create Gr-IDS (Staniford-Chen et al., 1996), a network-based IDS. GrIDS uses graph analysis of network activities for detection. In 1998, Steven Northcutt created SHADOW (Secondary Heuristic Analysis for Defensive Online Warfare) (Northcutt and Aldrich, 1998) IDS for the US Navy, as part of the CIDER[3] project. SHADOW performs traffic analysis using PERL scripts on SSH and tcpdump data, but suffers from high false positives and negatives. In 1997, SRI labs created a distributed hybrid IDS the "EMERALD" IDS (Porras and Neumann, 1997a). EMERALD is an extension to IDES/NIDES, which can detect network misuse, but suffered from high false positives and required manual creation of new rules for new attack/anomaly types. In 1998, Giovanni Vigna created "NETSTAT" (Vigna and Kemmerer, 1998), a NIDS that uses state-transition-analysis. Bro IDS (Paxson, 1999) was developed by Lawrence Berkeley National Laboratory in 1998. The same year Yoan created Prelude (Vandoorselaere, 1998), a hybrid IDS. In 1999, Martin Roesch developed SNORT (Roesch et al., 1999), an open source light weight intrusion detection system for networks. The IDS were now getting smarter and more network aware. However, they have limitations in the form of high false-positives and relatively easy and successful evasion by attackers. A compilation of evasion techniques is presented by Ptacek (1998)

IDS with Data-mining and Machine learning capabilities: At this point in time, with the increase in the number of detection sensors, especially due to distributed and hybrid IDS, researchers were faced with the problem of large datasets to process. This led to a shift towards detection methodologies that involve data mining and machine learning approaches. The same year Columbia university created a network-based misuse detection system called MADAM ID (Lee et al., 1999), an IDS that used data mining techniques. MADAM ID takes into account connection characteristics for analysis but does not consider packet payload data. Thus any attack having connection characteristics matching normal behaviour goes undetected. Mississippi state university developed an "Intelligent IDS (IIDS)" (Bridges and Vaughn, 2000), that used fuzzy data mining techniques. CERIAS developed "Autonomous Agents for Intrusion Detection (AAFID)" (Balasubramaniyan et al., 1998), a multi host-based distributed intrusion detection system. "ADAM or Audit Data and Mining" (Barbara et al., 2001) developed in 2001, employs data mining techniques on tcpdump audit trail for intrusion detection. Like MADAM, ADAM also uses connection-level data and does use the packet payload for analysis and detection. In 2001, the MITRE corporation developed an IDS called "Data Mining for Network Intrusion Detection" (Bloedorn et al.,

[3]"Cooperative Intrusion Detection Evaluation and Response"

2001). In 2002, Silicon Defence worked on an anomaly detection project called Statistical Packet Anomaly Detection Engine (SPADE) (Staniford et al., 2002), SPADE use packet header or connection data for statistical anomaly detection. "PHAD or Packet Header Anomaly Detection" (Mahoney and Chan, 2002), was developed by Mahoney et al. in 2004. An important limitation of this system is that It analyses features from packet header data for analysis and not the payload. Ertoz et al. (2004) developed "Minnesota Intrusion detection system (MINDS)", a network-based IDS that employs multiple modules for intrusion detection. MINDS also extracts multiple features from the packet header data for analysis and detection. Here it can be noted that although the sophistication and complexity of IDS has increased over the years; but still, most network-based techniques rely heavily on features extracted from packet header data alone. Over the years many extensions in the form of plugins have been added to these existing IDS. Computer attacks that can be exploited remotely over the network comprise mostly of known vulnerabilities, exploits and shell code, which are transported using the payload.

Intrusion detection techniques which ignore the payload data cannot act as effective defences and fail when attackers evade them by sending malicious payloads using headers resembling normal traffic or normal behaviour. Accordingly, in our work we have adopted techniques which go beyond examination of just the packet headers.

Payload-based Detection: Packet payload information gives an insight into application level data and is most useful in identifying attacks against vulnerable applications. For payload content-based detection, researchers have employed detection techniques and algorithms based on frequency distribution-based n-gram techniques. These techniques created models of normal network data and detected deviations from it. A few notable contributions include PayL (Wang and Stolfo, 2004) in 2004, McPAD (Perdisci et al., 2009) in 2009, and the recent work done by Miller (Miller and Hu, 2012; Miller et al., 2011). Some contributions that extend PayL by adding intelligence and unsupervised learning include (Bolzoni et al., April) and (Ariu and Giacinto, 2010). Such frequency distribution-based anomaly detection techniques are effective and better than header-based detection, but suffer from both high false positives and false negatives and require considerable memory and processing.

Information-theoretic Intrusion detection techniques: Other researchers found success with Kolmogorov Complexity and information-theoretic approaches for intrusion detection namely Lakhina et al. (2005); Feinstein et al. (2003); Wehner (2007);

Kulkarni and Bush (2001a); Evans and Barnett (2002); Evans et al. (2004); Kulkarni and Bush (2006); Eiland and Liebrock (2006); Flake (2004); Cilibrasi and Vitányi (2005); Leland et al. (1994). These techniques promise simple and effective methods for intrusion detection and an active area for research. Researchers have used these techniques to successfully detect focused problems such as: network scans, anomalous FTP and binary malware. However, there does not appear to be any generalized solution for malicious network streams.

Honeypots for Intrusion detection: Other researchers employed Honeypots to capture anomalous streams and generate signatures (Kreibich and Crowcroft, 2004; Mohammed et al., 2008; Pouget and Dacier, 2004; Levine et al., 2003; Julisch, 2002; Tang and Chen, 2007) (discussed in 2.1.6.5). Honeypots serve as an excellent data collection resource to capture malicious streams for analysis.

There has been a shift from generalized IDS to more focused intrusion detection techniques, such as those focusing on worm detection (Leland et al., 1994; Singh et al., 2004; Kreibich and Crowcroft, 2004; Kim and Karp, 2004; Newsome et al., 2005; Mohammed et al., 2008; Kim and Karp, 2004; Singh et al., 2004; Eiland and Liebrock, 2006; Kulkarni and Bush, 2006; Wehner, 2007; Evans and Barnett, 2002; Tang and Chen, 2007; Levine et al., 2003; Pouget and Dacier, 2004) (discussed in 2.1.6.5) and botnet detection (Gu et al., 2007).

This was a brief history and review of a few notable IDS, well known in the research community. More detail on these systems is presented in the subsequent sections. Many commercial products based on the aforementioned IDS and techniques have also been developed over the years, which are not listed here since they are the intellectual property of a company and their internal functionality is not fully disclosed or documented. A compiled time line of a few notable IDSs is illustrated in Figure 2.1.

2.1.2 A Taxonomy of Intrusion Detection Systems

In order to understand different classes of intrusion detection systems, the work done in each class and the gaps in existing intrusion detection techniques, it is helpful to study their taxonomy. Numerous Intrusion detection techniques have been proposed over the years. A comprehensive list of such resources is available on-line at the Center for Education and Research in Information Assurance (CERIAS), located at Purdue University CERIAS (2011). In 1998 DARPA created a working group called "The Common Intrusion Detection Framework (CIDF)", which was responsible for coordinating and defining a common framework in the IDS field. Later in 2000, this working group

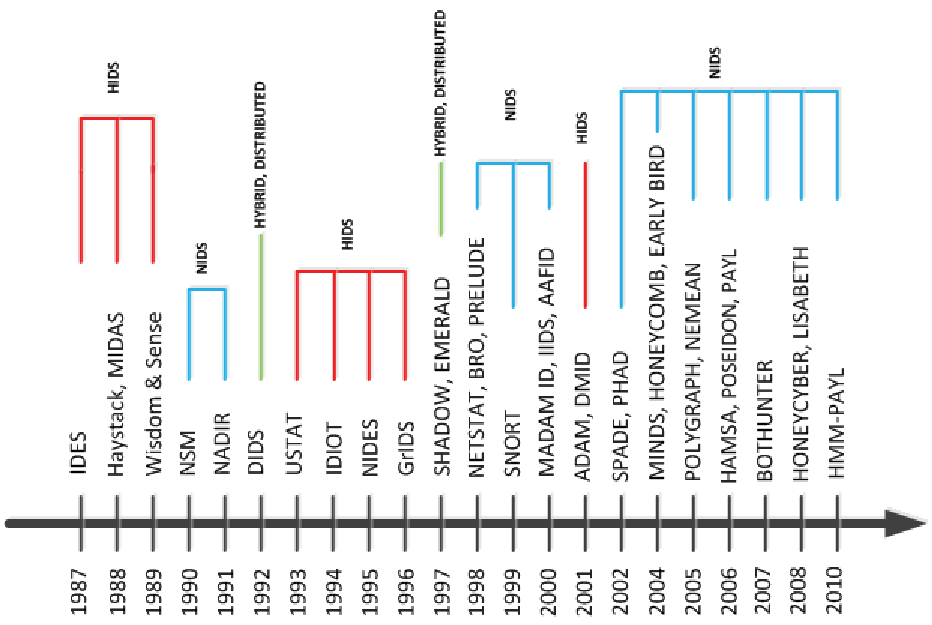


Figure 2.1: Time-line of a few notable Intrusion Detection Systems

was integrated within the "Internet Engineering Task Force (IETF)", under the new acronym IDWG, which stands for Intrusion Detection Working Group". Several classification schemes have been proposed by researchers in the past have proposed such as: Allen et al. (2000), Axelsson (2000), Debar et al. (1999), Jones et al. (2000), Kruegel and Toth (2000), Kvarnström (1999), Lunt (1993), Qing et al. (2004), Lazarevic et al. (2005), Sabahi and Movaghar (2008), Garcia-Teodoro et al. (2009), Amer and Hamilton Jr (2010) and Liao et al. (2012). However, unfortunately there is yet no universally accepted taxonomy for IDS. After careful study of these taxonomies, it was observed that at the core, all IDS are dependent on and can be distinguished by categories such as: information sources, analysis methods, location and detection methodologies. Other attributes of the IDS such as: architecture, time, and response are important but serve as extra categories to categorize IDSs. All these categories are important and highly integrated with each other. This strong coupling has troubled researchers in the past and they seemed to mix categories such as: information sources and location, though they are independent categories. Based on these and other observations and shortcomings of previous taxonomies, it was decided to group IDSs using the following seven categories as a criteria. This is a general taxonomy that is conceived by adding, splitting and merging several categories from previous taxonomies, mostly extracted from Lazarevic et al. (2005), Sabahi and Movaghar (2008), Amer and Hamilton Jr (2010) and Liao et al. (2012):

1. Analysis Method

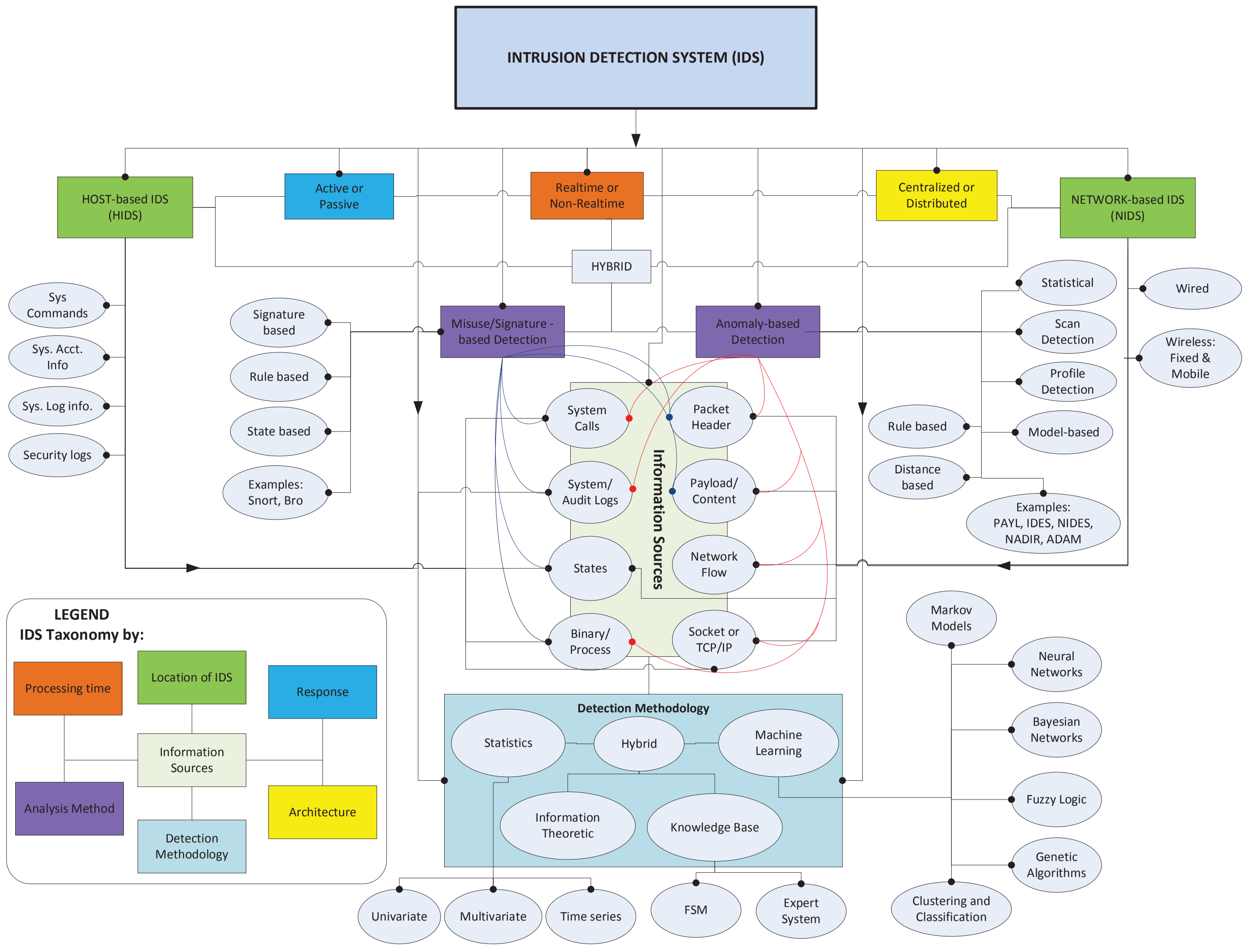


Figure 2.2: Taxonomy of Intrusion Detection Systems as per the proposed seven category criteria

2. Location in the Network
3. Architecture
4. Processing time
5. Response
6. Information Sources
7. Detection Methodology

This proposed taxonomy is also illustrated graphically in Figure 2.2, here the relationships are also highlighted between different categories. A summary of these categories is also illustrated in table 2.1

2.1.3 Categorization by Analysis Method

CERIAS classifies IDS under the two most widely used categories, which are: *Misuse detection* based and *Anomaly detection* based.

Researchers in intrusion detection have used many different techniques, but these can broadly be classified into the following categories and sub-categories:

- Misuse-based intrusion detection: Performs predefined signature or pattern matches on events or strings.
  - Signature Based.
  - Rule Based.
  - State Based.
- Anomaly-based intrusion detection: Detection on observing deviations from normal behaviour of the system. May use statistics, information-theoretic and machine learning techniques for detection.
  - Statistical methods.
  - Scan detection: Focused on detecting network scans.
  - Profile detection: Aggregate network connections to identify behavioural features, often using clustering techniques.
- Hybrid detection: Uses the combination of multiple techniques for detection.

IDS Taxonomy Summary		
Category	**Main Attributes**	**Examples**
Analysis Method	Misuse-based	Snort, Bro, STAT, NetSTAT, Haystack
	Anomaly-based	IDES, NIDES, ADAM, MADAM, EMERALD, SPADE, MINDS, PHAD
Location	Network-based	Snort, Bro, NSM
	Host-based	IDES, MIDAS, Haystack, W&S, OSSEC, BSM, LSM, SELinux
Processing Time	Realtime or Online	Snort
	Non-realtime or Offline	Argus, DIDS
Response	Active or Pro-active	Configuration Dependent
	Passive or Reactive	Configuration Dependent
Architecture	Centralized	Snort, Bro, STAT, OSSEC
	Distributed	DIDS, GrIDS, Argus
Information Source	System calls, Audit trail, Logs, Socket/tcpip data, Network flow, Packet header info., Packet payload, States, Process/binary	Snort, Bro, STAT, NetSTAT, Haystack, IDES, NIDES, ADAM, MADAM, EMERALD, SPADE, MINDS, PHAD, OSSEC, BSM, LSM, SELinux, DIDS, GrIDS, Argus
Detection Meth.	Statistical	IDES, SPADE, PHAD
	Data-mining and Machine Learning	MADAM ID, IDDS, ADAM, NIDES, Lee et al. (1999), Sung and Mukkamala (2003), Kruegel and Toth (2003), Chebrolu et al. (2005), Liu et al. (2002), Chen et al. (2005),Liao and Vemuri (2002), Wang et al. (2006), Soule et al. (2005), Portnoy et al. (2001), Zhang et al. (2008) and others
	Information theoretic	Lakhina et al. Lakhina et al. (2005) , Wehner Wehner (2007), Kulkarni and Bush Kulkarni and Bush (2001a), Evans and Barnett Evans and Barnett (2002), Kulkarni, Evans and Barnett Kulkarni and Bush (2006), Eiland and Liebrock Eiland and Liebrock (2006), Halvar Flake Flake (2004),

Table 2.1: Proposed IDS Taxonomy

2.1.3.1 Misuse-based Detection Systems:

Misuse or signature based detection is used to identify unique patterns of unauthorized activity, and use these patterns to predict and detect subsequent similar attempts.

Notable Misuse detection systems include: Snort (Roesch et al., 2006), Bro, Haystack, STAT, MIDAS and IDES.

Misuse detection relies on already available knowledge of an attack or exploit in the form of a signature. The signature creation process may require extensive analysis of an attack by a human expert. This may increase the accuracy of a misuse-based IDS, however they may be unable to cope with attack variants, and novel and emerging threats, as they lack the intelligence to detect them and any variations. Misuse-based IDS may incorporate the following methods for detection:

- Signature-based detection.
- Rule-based detection.
- State-based detection.
- Data-mining methods for detection.

Signature-based detection involves matching monitored events to known attacks, by searching through a signature database of known attacks for each event. If a monitored event matches a known attack, alarms are raised or action is taken by the system. This detection methodology, however, is not capable of detecting novel or emerging threats. Signatures require careful analysis by a human expert to be created, while an exploit is actively exploiting vulnerable systems over the network. Once a signature is created and tested, it is then added to the signature database for use. Examples of well-known systems that incorporate signature-based detection methodology include: Snort (Roesch et al., 2006), Haystack (Smaha, 1988) and others.

Rule-based detection methods These use rules to identify attacks. Rules are created using a rule-based language to describe characteristics of an attack. These rules are then applied on the collected data to detect attacks or intrusive patterns. Examples of rule-based IDS include Bro (Paxson, 1999), which uses a rule based language to create rules and policies that define intrusive activity. IDES (Intrusion Detection Expert System) Javitz and Valdes (1991), developed by SRI International. This system had a rule-based expert system along with a statistical anomaly detection sub-system. EMERALD IDS (Porras and Neumann, 1997a) is another rule-based IDS. It uses the

Production Based Expert System Toolset (P-BEST) (Lindqvist and Porras, 1999) which is a rule-based expert system developed at SRI. P-BEST is also used as the rule-based inference engine of NIDES (Next-gen Intrusion Detection Expert System) (Anderson et al., 1995).

State-based detection State-based detection methodologies involve detecting intrusive system states. The system is modelled as a finite state machine. Malicious activities or attack scenarios are modelled as states. This serves as a reference malicious state. System states transitions are then monitored and any state that resembles the malicious reference state is flagged as a threat or intrusion. State-based detection can be modelled on a host's security audit trail. Examples of such systems include USTAT (Unix State Transition Analysis Tool) (Ilgun, 1993), based on STAT (Porras and Kemmerer, 1992). State-based detection can also be modelled using network data, as done by NetSTAT (Network-based State Transition Analaysis Tool) (Vigna and Kemmerer, 1998). In such systems attack scenarios (modelled as state transition diagrams) and the network itself are modelled as hypergraphs, with network interfaces represented as nodes and hosts modelled as edges on the graph. This results in a highly customizable architecture for detecting attacks and probes, as it has an extra advantage that network topological knowledge added to the system.

Data-mining and machine learning methods for misuse detection imply learning algorithms on benign and intrusive labelled data to train the system. Once the system gets trained, it can then be used on live system or network data for misuse detection. A comprehensive account of techniques that use data-mining and machine learning are discussed in Section 2.1.6.3

2.1.3.2 Anomaly-based Detection Systems:

Anomaly detection systems raise alarms on detecting significant deviations from normal patterns or models. An anomaly is defined as a deviation from the normal or expected behaviour or outcome of a system.

A few notable anomaly detection systems include: Snort with Spade plugin, MADM, ADAM, NIDES, and EMERALD.

Anomaly-base detection has the capability to detect novel and emerging threats, but may incorporate numbers of high false positives. Researchers in the past have used many different algorithms for anomaly detection systems. Anomaly detection techniques mainly differ in terms of source of information and methodology. Anomaly-based detection techniques can be classified into the following main groups:

1. Statistical Methods
2. Rule-based methods
3. Distance-based methods
4. Profiling methods
5. Model-based methods

Statistical Methods This technique involves applying statistical methods to detect intrusions in system and network data. This is done by extracting features from data and translating them into variables. Next, statistical techniques such as average, standard deviation or others are used to detect abnormal deviations in these variables. Such simplistic statistical models can be applied to host and network based IDS. Most anomaly detection systems depend on statistical features extracted from the network traffic such as: source and destination ip, source and destination port, latency changes, arrival rates, traffic volume. Examples of anomaly-based systems that use statistical methods include: IDES (Javitz and Valdes, 1991), NIDES (Anderson et al., 1995), EMERALD (Porras and Neumann, 1997b) and SPADE (Staniford et al., 2002). SPADE is available as a plugin for SNORT (Roesch et al., 2006) and can detect port scans using probability and frequency techniques. Other systems use probability, chi-square statistics, outlier detection and others.

Rule-based Methods In anomaly detection systems, rule-based methods are used to create rules to define ‘normal’ behaviour in system and network data. Any deviation from these rules is considered an anomaly and flagged for inspection by an expert.

Distance-based Methods To overcome the statistical limitations posed by multi-dimensional data, distance-based methods are used to identify outliers by calculating distances between all data points from one another and then calculating their neighbourhood densities. Lazarevick et al. has given a good comparison of several distance-based methods in their technical report (Lazarevic et al., 2003). A good example of a distance-based system is MINDS (Minnesota Intrusion detection system) (Ertoz et al., 2004). This system uses net-flow data to extract features. All network connections are assigned an anomaly score, which is then investigated by an expert.

Profiling Methods Profiling methods involve creation of normal user, system and network behaviour profiles. Any deviation from these profiles is considered as intrusive behaviour. They incorporate several data-mining and heuristics for decision making. Examples of such system include ADAM or Audit Data and Mining (Barbara et al., 2001), that uses data-mining to detect attacks in tcpdump data. PHAD or Packet Header Anomaly Detection (Mahoney and Chan, 2002) is yet another IDS of this type. PHAD is capable of extracting multiple (up to 33) features from packet headers to build behaviour profiles. These profiles are then used for anomaly detection. There are some other systems, such as those that are inspired by the human immune system, and those that perform behaviour modelling using association pattern analysis or association rules, rate-based profiling methods and connection based profiling methods.

Model-based Methods Anomaly-based IDS mostly employ model-based methods (ADAM, NIDES, EMERALD, PayL). Using model-based methods, normal and anomalous behaviour models are created for training the system. Once trained, the system can identify intrusions based on deviations from normal behavioural models and similarity with anomalous behaviour. Researchers have used supervised and unsupervised learning techniques for model-based anomalous intrusion detection. Commonly used techniques include Markov-model[4], Artificial neural networks (ANN)[5] and k-NN (k-nearest neighbour) based approaches.

For our proposed IDS, we employ multiple anomaly-based detection techniques to enhance detection. We employ profiling methods to create profiles of packet or stream payload data in our work, distance-based methods to determine a similarity distance between these profiles, model-based methods to create models to guide the classification.

2.1.3.3 Hybrid Intrusion Detection Systems:

Hybrid detection systems incorporate multiple detection schemes to provide extensive and accurate detection. This may include a combination of network-based and host-based approaches and signature-based and anomaly-based approaches and their variants. Hybrid systems often correlate information and events received from multiple sensors to identify intrusions. This is done by using multiple correlation and data-mining approaches. Examples of such systems include the distributed intrusion detection system (DIDS), developed by Snapp et al. (1991). This system makes use

[4]`http://www.mathpages.com/home/kmath232/part2/part2.htm`
[5]`http://en.wikipedia.org/wiki/Artificial_neural_network`

of a HIDS called Haystack (Smaha, 1988) to detect local attacks, while network security monitor or (NSM) (Heberlein et al., 1990) is used to monitor the network. Both systems, Haystack and NSM, send information to the DIDS Director, where the final analysis is performed.

2.1.4 Categorization by Location in the network:

Generally, intrusion detection systems can be categorized into one of two main categories, having sub-categories. This is based on their location or deployment in the network:

- Host-based IDS (HIDS): Uses System Commands, System Accounting info, System Log information and Security Logs
- Network-based IDS (NIDS)
  - Wired network
  - Wireless network
    - Mobile
    - Fixed

2.1.4.1 Host-based IDS (HIDS)

Early IDS emerged as HIDS deployed on Unix systems (discussed in 2.1.1. Host-based IDS monitor and analyze user activity on a host system. They are able to derive first-hand data from attackers interacting with the host system. Since HIDS are actively monitoring the host system, they can be detected by the attackers and in some extreme cases get tampered with or even shut down by them. It is imperative for such systems to quickly report intrusive behaviour to a remote server before they get spotted and turned off or tampered with by an attacker. They can provide information sources such as system commands typed on the console by the user, system accounting information, process logging information, and security audit information.

System Commands can be used to detect malicious activity by analyzing commands typed by the user. Since malicious users may use a different set of commands from ordinary users, they can be profiled as having malicious or intrusive intent.

System Accounting information such as process accounting (pacct file in Unix), which contains information about a process owner, execution time and memory, and login accounting information (wtmp and access.log file in Unix), which contains information about successful login and logout along with unsuccessful login attempts, are a major source of accounting information. They can be used to detect attacks or intrusive patterns.

System Log information or syslog provides information that is logged by both, the system and applications or services. Hence any interaction of the malicious user with the system and or its applications and services would be recorded by syslog. Syslog is highly configurable and can be setup to speak with many Unix services such as 'login', 'httpd', 'sendmail', 'nfs', 'ftpd', 'mysqld', 'sudo', 'swatch', 'TkLogger' and others. The examples here relate to Unix/Linux specific services, but similar methods are applicable to other operating systems like Windows and Mac OSX.

Security Logs on a host contain audit logs generated by the host security module. This module is responsible for maintaining and logging a chronological list of security events that have taken place on a host. This information is collected by inserting hooks into the kernel that detect all user-level system calls interfaces with the kernel. This information may include authentication information, access control information, system call traces, user and group information, CPU and memory allocation and file reads. Analysis of these logs may reveal intrusive activity or patterns. In Sun Solaris systems, The Basic Security Module or BSM (Soft, 1995) is responsible for monitoring and recording security related events. For Linux systems the Linux Security Module or LSM (Wright et al., 2002) performed the same task, which is now replaced by the SELinux (Smalley et al., 2001) module.

2.1.4.2 Network-based Intrusion Detection Systems or (NIDS)

They monitor network traffic to detect intrusive behaviour. With the exponential growth of computer networks and services available on the Internet, the need to monitor and safeguard network protocols, segments and devices has become imperative. A detailed discussion of NIDS systems has been presented earlier (section 2.1.1).

NIDS can be deployed at strategic locations in the network, mainly at entry and exit points or network gateways. NIDS can detect various known attacks and intrusive behaviours at the network level by analyzing packets flowing through its sensors. Packet information may include header and payload information. Complete communication between hosts can be retrieved and studied using tcpstream-reassembly or packet reassembly. Packets are usually collected by a sniffer sub-system, which copies packet instances from network interfaces for analysis. They can provide insight into low level packet details and protocol structures.

2.1.5 Categorization by Information source

Another interesting way to classify intrusion detection systems is by their information or input sources as:

- Header-based detection
- Payload/content-based detection
- Hybrid approach
- System calls
- Event log
- TCP/IP data
- Binary/executable analysis
- Network flow aggregation

Intrusion detection systems initially emerged as host-based systems. As users logged into mainframes for their computational tasks, the security and monitoring of user activity on these systems became a requirement. Hence, information sources were limited to host-only events. However, with the exponential growth of computer networks, network data became an information source for IDS. Today many systems focus on either or both in a hybrid arrangement.

2.1.6 Categorization by Detection Methodology:

Due to the vast amount of data generated by various distributed host and network-based detection sensors, there is a requirement for a capability to quickly and efficiently parse these large datasets and correlate information between them to identify patterns and make intelligible decisions. This may involve creating models after discovering knowledge and patterns from datasets and using learning algorithms to learn and predict changes intelligibly and suggest responses automatically.

Recent trends in IDS research suggest a shift from misuse and anomaly based methodologies for intrusion detection towards those that incorporate data mining and statistical machine learning techniques (as discussed in section 2.1.1). They can be categorized as:

- Statistical method based detection: This may include Univariate, Multivariate, Time series model.
- Knowledge-based detection: This includes Finite state machine (FSM) and Expert Systems.
- Machine learning based detection: This includes Bayesian Networks, Fuzzy Logic, Neural Networks, Markov models, Genetic Algorithms, Clustering and Classification.
- Information theoretic based detection.

2.1.6.1 Statistical Methods for Intrusion Detection

Research on Intrusion Detection Systems can be categorized by the use of statistical techniques and methods used for intrusion detection. This may include techniques involving mean, standard deviation, probability, Chi-square statistics and others. A detailed discussion on statistical techniques has been discussed earlier in the section on "Anomaly Detection" (See: 2.1.3.2)

2.1.6.2 Knowledge-based detection

Such systems are focused on availability of prior knowledge or data. The expert system approach is a well known knowledge-based IDS scheme, in which decision making is modelled around the behaviour of human experts. This is done by identifying different groups, classes, parameters and procedures. Next, rules are created around these, which help in data classification. Finite state machines are also used under knowledge-based techniques, in which states and transitions are analysed and decisions are taken based on them.

2.1.6.3 Machine learning approaches for Intrusion Detection

Machine learning is a branch of Artificial Intelligence (AI), that deals with the study and creation of evolving systems or algorithms that can learn and evolve from data. A machine learner is trained on labelled data and can then use that knowledge to identify patterns in unlabelled data for classification, and identify new patterns as they emerge. Machine learning algorithms are now being widely deployed for intrusion detection. Machine learning algorithms can be categorized as:

1. Supervised learning [6]
2. Unsupervised learning [7].
3. Semi-supervised learning [8]

Generally machine learning algorithms may be mapped to the following important categories:

- Fuzzy Logic[9]
- Markov Model[10]
- Bayesian Networks[11]
- Genetic Algorithms[12]
- Clustering and Classification[13]

Many machine learning algorithms are now being used for intrusion detection. A few notable techniques are mentioned below:

[6]A machine learner or classifier is trained on a labelled dataset of examples. It uses a function to map samples to correct classes, by utilizing prior knowledge from the training phase. It is often referred to as a classification problem. Examples include: Artificial Neural Networks (ANN), Bayesian statistics, Kernel estimators, Support vector machines (SVM), Random forest and the Nearest neighbour algorithm.

[7]The machine learner has no prior knowledge of the data structure and attempts to identify hidden structures in such unlabelled data. It is similar to density estimation in statistics and often referred to as a clustering problem. Examples include: k-means, hierarchical clustering and self-organizing maps (SOM)

[8]These learning methods use both supervised and unsupervised learning techniques in a hybrid manner for better classification.

[9]Use of fuzzy predicate logic to yield an approximation.

[10]Markov models comprise of interconnected states and derives a model by studying transition between states. Mainly comprise of markov chains and hidden markov models. Markov model based techniques have been used by researchers for intrusion detection.

[11]Bayesian networks use probability to encode relationships between variables. In intrusion detection they can be used to detect intrusive behaviour by analysing the probability of system or network events.

[12]Evolutionary algorithms inspired by biology. Used in IDS as they provide robust global search methods.

[13]These algorithms group or cluster similar items together, based on proximity from a reference point, density points or neighbouring points. Clustering algorithms are now being widely adopted by IDS.

Misuse detection using Statistical Machine learning Methodology Lee et al. (1999) applied association rules in audit data and network traffic for misuse detection. Sung and Mukkamala (2003) applied SVM on host and network logs to identify attacks and misuse patterns causing computer security breaches. Kruegel and Toth (2003) designed an algorithm to generate a decision tree for fast detection of malicious events from network data. Chebrolu et al. (2005) applied the CART-tree algorithm on the KDD cup 1999 intrusion detection dataset and eliminated features that did not contribute to the ranking, which increased overall accuracy of the system. They also investigated a feature selection and classification algorithm involving Bayesian networks on host data (Chebrolu et al., 2005). Most of the work done using these algorithms focuses on data obtained from host-based events or features extracted from packet headers for network-based data. These techniques do not consider network payload, hence cannot reliably determine the attack without payload analysis.

Anomaly-based detection using Statistical Machine learning Methodology Liu et al. (2002) used artificial neural network (ANN) techniques to analyze sequences of system calls for anomaly detection with good accuracy. Chen et al. (2005) compared SVM and ANN on 1999 DARPA evaluation set and BSM audit data and found that SVM outperformed ANN. Liao and Vemuri (2002) used KNN to classify program behaviours as normal or intrusive based on their system calls. For their dataset they observed a high accuracy with very low false positive rate. Wang et al. (2006) applied HMM to host audit data to detect anomaly data quickly at a lower mismatch rate. However the training phase requires multiple passes, resulting in slow detection rates and limiting the detection to host-based systems only. Soule et al. (2005) used Kalman filters to recognize traffic patterns using a network-wide view. Portnoy et al. (2001) applied clustering anomaly detection methods on both DARPA and KDD cup 1999 datasets and obtained good results in terms of classifier accuracy for known or labelled data, whereas, clustering unlabelled data resulted in lower detection. Zhang et al. (2008) applied the random forest algorithm to the KDD cup and DARPA datasets. They observed comparatively better results than other unsupervised techniques, however the classifiers performance became degraded over minority attacks. The algorithms and methodologies discussed in this section also rely on features extracted from packet headers and do not consider network payload for analysis.

Statistical machine learning algorithms are used in this thesis for learning known exemplars for classification and unknown exemplars for novelty detection and classification.

2.1.6.4 Information-theoretic approaches for Intrusion Detection

This section outlines information-theoretic approaches that are being used for intrusion detection.

Information-theoretic approaches for intrusion detection have recently been explored by researchers. Lakhina et al. (2005) demonstrated the detection of a wide range of anomalous network flows by examining their entropy measures. Feinstein et al. (2003) identified DDoS attacks by examining the entropy of packet header fields. Information-theoretic approaches can also be applied to packet or stream payload data for measurements and analysis. A few notable techniques used by researchers are discussed here: Wehner (2007) created a fast method for guessing the family of an observed worm without disassembly, by comparing the compressibility amongst binaries to determine their similarity using Kolmogorov Complexity (KC). For network traffic Wehner (2007) used average compression ratios of good or expected traffic and flagged anomalies on deviations from it. There has been a significant amount of work on network traffic characterization but little has been done in compression-based clustering and classification for traffic. This can be because compression is an expensive operation in terms of CPU time. Kulkarni and Bush (2001a) attempted a similar approach to monitor network traffic without using compression. Work carried out by Evans and Barnett (2002) used compression to compare the complexity of legal FTP traffic with illegal traffic, but at a header level only, whereas we shall use entire TCP payload sessions.

Evans and Barnet gave the concept of “conservation of complexity” (Evans and Barnett, 2002). They asserted that the Kolmogorov Complexity (KC) can be used as a fundamental property of information to build models and laws of information security. They applied this concept to network protocols to identify normal or attack traffic. Their initial experiment focused on transport headers of FTP sessions, using Unix Compress utility. Attack and normal traffic dumps were converted to ASCII first and then compressed incoming and outgoing traffic sets, while removing noise such as header information. They found that FTP has predictable complexity. Their preliminary work focused only on TCP session data and not the payload. Later Evans, Eiland and Markham worked on a scheme that used the ASCII text stream of the FTP control channel (Evans et al., 2004) and got comparatively better results. Kulkarni et al. Worked on DDOS detection using Kolmogorov Complexity, based on the theme that “Highly correlated data has a high compression ratio” (Kulkarni and Bush, 2006). Since in a DDOS attack similar packets are sent to a single target from distributed locations, their similarity can be used to detect such attacks. Packets are sampled from flows and their complexity differential is calculated. Highly correlated packets indicate existence of a pattern for detection. Kulkarni and Bush (2006) performed denial of service measures using compression ratios based on Kolmogorov Complexity by estimating the entropy of 1's contained in the packet. Such techniques generally prove to be quite efficient in determining worm and worm-like activity, but suffer from limitations that may lead to false positives. Eiland and Liebrock (2006) detected network scans from normal traffic using Kolmogorov Complexity by estimating the inverse compression ratio of scan and normal traffic. This technique fails when used to partition or classify various types of attack traffic using full payload data.

Flake (2004) and Carrera and Erdlyi compared executable objects by instigating graph-based methodologies. Flake (2004) also applied this technique to the analysis of malware by disassembling the binary. Wehner (2007) created a fast method for guessing the family of an observed worm without disassembly, by comparing the compressibility amongst binaries to determine their similarity. There has been a significant amount of work on network traffic characterization but little has been done in compression-based clustering and classification for traffic, possibly because compression is an expensive operation. Wehner (2007) employed Kolmogorov's Complexity using a method suggested by Cilibrasi and Vitányi (2005) to determine similarities. Kulkarni and Bush (2006) attempted a similar approach to monitor network traffic without using compression. Work carried out by Evans and Barnett (2002) used compression to compare the complexity of legal FTP traffic with illegal traffic. Evans and Barnett (2002) performed denial of service measures using compression ratios based on Kolmogorov complexity. Such techniques are effective, but limited to focused attacks like FTP or scan traffic. We explore these techniques further and propose more general solutions to enhance detection.

In this thesis, such techniques are applied to packet or stream payload data, as a method to determine similarity between them.

2.1.6.5 Worm Detection Systems

Due to the fast propagation techniques, sophistication and large scale havoc caused by computer worms, they have recently become an interesting problem for research and security researchers. In the last two decades, many worm detection techniques have emerged and considered as an important capability from an IDS perspective. A few notable worm detection techniques are discussed here: The existence of complex self-similar patterns in Internet traffic was first revealed in work done by Leland et al. (1994). Multiple invariant sub-strings must often be present in all variants of a worm payload (Leland et al., 1994). Generation of a short single substring signature for every worm instance can result in a high number of false positive rates (Leland et al., 1994). Systems based on pattern-based analysis extract common byte patterns across suspicious flows to generate a signature for novel Internet worms. Examples of such systems include EarlyBird (Singh et al., 2004), Honeycomb (Kreibich and Crowcroft, 2004), and Autograph (Kim and Karp, 2004). A single signature is used to match all worm instances based on unique sub-strings in the payload. These sub-strings are considered invariant across worm connections. Such systems may suffer from a relatively high false positive and high false negative rate. Earlybird, Autograph and Honeycomb generate a single, contiguous substring of a worm's payload as a signature. This inhibits their ability to detect all polymorphic worm instances.

Classification of signatures for polymorphic worms are usually divided into two main categories: (Newsome et al., 2005; Mohammed et al., 2008; Kim and Karp, 2004; Singh et al., 2004):

- Content-based: Detect similarity in different instances of byte sequences to characterize a given worm.
- Behaviour-based: Characterization by perceiving the semantics of byte sequences. (Execution behaviour.)

The data from Honeypots can provide some insight information useful for intrusion and attack analysis. Here we discuss systems that generate offline signatures for any instance of a polymorphic worm. Pouget and Dacier (2004) analysed traffic in Honeypots to identify root causes of frequent processes. Levin et al. explained the use of Honeypots in extracting particulars of a worm (Levine et al., 2003). Julisch (2002) defined a method that clustered intrusion alarms for discovering the root cause of an alarm. Content-based systems like Honeycyber, Polygraph, Hamsa and LISABETH (Newsome et al., 2005; Li et al., 2006; Cavallaro et al., 2008; Mohammed et al., 2008) generate automated signatures for polymorphic worms. The commonality between these systems is their use of several distinct sub-strings that are often present in variants of polymorphic worm payloads regardless of whether or not the payload changes for each infection.

All these systems capture packets from a gateway router, thus these systems may find multiple polymorphic worms addressing a different vulnerability from the other. This can make it difficult to find distinct content shared amongst polymorphic worms. To capture most polymorphic worm instances, it is necessary to observe the polymorphic worm while it interacts with the host. Honeycyber (Mohammed et al., 2008) utilizes a "Double-Honeynet" method to detect polymorphic worms and collect all their instances. Sommer and Paxson (2003) proposed adding connection-level context to signatures to reduce false positives. Yegneswaran et al. (2005) described the Nemean system, that incorporates protocol semantics into the signature generation algorithm, giving it a wider coverage in dealing with polymorphic worms. Tang and Chen (2007) defined a "Double-Honeypots" system. Since polymorphic worms use random keys to encrypt each instance of it, this will result in huge number of signatures that would increase with every iteration or propagation. Since after decryption of the polymorphic function, the op-code remains consistent, a better approach would be to develop a signature for the in-memory run-time op-code used by the polymorphic worm.

Information-theoretic approaches for worm detection have been successful. Work done by Wehner (2007), Kulkarni and Bush (2001b),Kulkarni and Bush (2006), Evans and Barnett (2002) and Eiland and Liebrock (2006) has been discussed in detail in the section 2.1.6.4

2.1.7 Categorization by Processing time:

IDS can be categorized by the time they take to process data, making them online or 'real-time' IDS, in which data is processed in parallel or almost simultaneously on reception. IDS can also be offline or 'non real-time' IDS, in which data is first collected and later processed in batch mode. Due to the extensive processing and learning of malicious streams, the IDS proposed in this thesis is a non real-time IDS.

2.1.8 Categorization by Architecture:

From architectural view point, IDSes can be 'Centralized', i.e. they monitor a single host or server and analyse its data only. They can also be 'Distributed', i.e. they can monitor several distributed hosts or sensors and collect, consolidate and analyse their data to detect global, distributed and coordinated attacks and patterns. Distributed architecture may give better insight into attacks, as they have the capability to coordinate cyber attacks from several distributed locations. Distributed IDSes often employ agents on remote hosts to collect and exchange data. The data is then correlated by an inference engine to detect attacks. This process may incorporate methods based on similarity in alert features such as source or destination IP address and ports, it can also decide based on correlation of known attacks scenarios, which need to be specified prior to detection. Another way to detect attacks is by correlating pre and post attack data. Well-known examples of Distributed IDS include: DIDS (Snapp et al., 1991), GrIDS (Staniford-Chen et al., 1996) and Argus (**?**). The IDS proposed in this thesis can be employed as both centralized, on a single host and distributed, by collecting data from multiple hosts or Honeypot sensors.

2.1.9 Categorization by Response:

IDS can be categorized by their response to attacks or intrusions. This response can be 'active', which can further be 'pro-active' i.e. on detection of intrusive behaviour the IDS can take pre-emptive countermeasures such as lockout the attacker and kill suspicious processes run by him and in some extreme cases, launch an automated attack against the attacker, or 'reactive', i.e. on detection of intrusive activity it can react to the incident in a prescribed way, defined by the administrator. An IDS should actively detect and respond to attacks as they occur. Conversely, an IDS can be 'passive'. In this scenario, on detection of malicious activity, the IDS would passively log or drop malicious packets. A response may be active or passive, but either way maximum information about the attack and the attacker should be collected by the system. The IDS proposed in this thesis can act actively on detection of known malicious streams and passively on detection of new or novel malicious or suspicious streams.

2.1.10 Gaps in Intrusion Detection Systems

There has been a significant amount of work done on intrusion detection technologies, the most notable of which has been reviewed in this chapter. The literature has demonstrated a clear shift from traditional statistical and pattern matching intrusion detection schemes to hybrid data mining and machine learning approaches. Techniques involving information-theoretic measures have shown promising result and have recently emerged as more simple techniques for intrusion detection. For NIDS, it has been observed that most techniques rely on features extracted from packet header data. Such techniques become unreliable and easy to evade (Ptacek, 1998). Researchers are now employing techniques that involve payload analysis.

From the literature review we conclude that applying statistical techniques to packet header information alone cannot serve as a reliable intrusion detection technique, as most of the information can be extracted from the payload part of a packet or TCP stream. Many evasion techniques (Ptacek, 1998) can be used to undermine IDS that solely rely on packet header information. We assert that the information present inside the packet payload is unique to an attack or intrusion and hence can be used to classify attacks. Information-theoretic measures can be applied to measure this payload information and classify attacks accordingly. Some approaches that use similar techniques have been discussed in section 2.1.6. The intrusion detection techniques proposed in this thesis are anomaly-based detection schemes with an information-theoretic focus, that incorporates multiple techniques such as: profiling methods, distance and model based methods using statistical machine learning algorithms for detection.

2.2 Classifying Computer Attacks

This section defines and describes different types of computer attacks and presents a review of their taxonomy. Knowledge of these attacks is useful to both the attackers to infiltrate computer networks and systems, and system administrators or defenders or security engineers to safeguard against the attackers. For our thesis we use this knowledge of attacks to label attack samples in order to create training sets to train our proposed IDS.

The Committee on National Security Systems (CNSS) (on National Security Systems, 2003) of the United States of America defines a computer attack as:

"Any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself."

2.2.1 Types of Attacks

A comprehensive list of computer attack definitions are listed in resources like the Internet Security Glossary (RFC 2828) (Shirey, 2000). Generally an attack can be divided into two main types, based on an attacker's response:

- Active: "An active attack attempts to alter system resources or affect their operation" (Shirey, 2000). Examples may include: Denial-of-service (DOS)[14] attack, Spoofing, Man-in-the-middle (MITM)[15] attack, ARP poisoning, Ping flood, Ping of death[16], Smurf attack[17], Buffer overflow[18], Heap overflow and Web-based attacks.
- Passive: "A passive attack attempts to learn or make use of information from the system but does not affect system resources" (Shirey, 2000). Examples of such types of attacks include: Sniffing[19] and Wiretapping.

Based on source or origination, an attack can further be divided into:

[14] "The prevention of authorized access to a system resource" (Shirey, 2000).

[15] "A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association." (Shirey, 2000)

[16] "An attack that sends an improperly large ICMP echo request packet (a ping) with the intent of overflowing the input buffers of the destination machine and causing it to crash." (Shirey, 2000)

[17] "Software that mounts a denial-of-service attack (smurfing) by exploiting IP broadcast addressing and ICMP ping packets to cause flooding." (Shirey, 2000)

[18] "A process that gains control or crashes another process by overflowing the other processs buffer." (Hansman and Hunt, 2005)

[19] "A synonym for passive wiretapping" (Shirey, 2000)

- Insider: "An inside attack is an attack initiated by an entity inside the security perimeter (an insider), i.e., an entity that is authorized to access system resources but uses them in a way not approved by those who granted the authorization" (Shirey, 2000).
- Outsider: "An outside attack is initiated from outside the perimeter, by an unauthorized or illegitimate user of the system (an outsider). In the Internet, potential outside attackers range from amateur pranksters to organized criminals, international terrorists, and hostile governments" (Shirey, 2000).

Based on medium or target, an attack can be:

- Network-based: Attacks that target, damage or influence a hosts network connectivity. Examples include: Denial-of-service (DOS) attack, Spoofing or masquerade attacks, Man-in-the-middle (MITM) attack, ARP poisoning, Ping flood, Ping of death, Smurf attacks and others.
- Host-based: Attacks that target, damage or influence a host system. Examples include: Buffer Overflows, Heap Overflows and others

2.2.2 Why is it important to understand Computer Attacks?

In the intrusion detection literature a computer attack may encompass all means that can be used in an intrusion attempt. Thus a successful computer attack may lead to an intrusion (Powell and Stroud, 2001). It is important to understand and categorize attacks, as this may lead to better defences and detection. For our thesis we study computer attack taxonomies to develop an understanding of the attacks and their categories and use this knowledge to assign labels when creating a dataset for IDS evaluation.

2.2.3 A Brief Taxonomy Of Computer Attacks

Attack classification has always been an interesting area for security researchers. Over the years, many classification techniques have been proposed and adopted and later replaced by better techniques. MITRE's Common Vulnerabilities and Exposures (CVE) (MITRE, 2012) list provides a common name of all publicly known attacks. This list or dictionary, however, is not a taxonomy. The CVE list contains a unique identifier and a description of the vulnerability, along with references. Researchers have also attempted to use attack patterns ((Hoglund and McGraw, 2004), (Moore et al., 2001)), derived from design patterns, which are used to describe the attackers perspective of an attack to classify attacks. MITRE's common attack pattern enumeration and classification (CAPEC) is a community developed resource of attack patterns. Although it provides a good means for taxonomy, however it suffers from some subjective categories that are misleading and may cause confusion.

A good computer attack taxonomy should be able to describe and classify an attack without ambiguities or repetitions. In the past, researchers have attempted to categorize computer attacks and intrusion on the basis of vulnerabilities (Landwehr et al., 1994; Lindqvist and Jonsson, 1997; Lough, 2001; Bisbey and Hollingworth, 1978; Abbott et al., 1976). Other taxonomies focus on attacks, their perpetrators and the result (Lazarevic et al., 2005). A comprehensive list of computer attacks, comprising of 3000 documented attacks and misuse in the form of a Computer Abuse Method Model (Neumann, 1994; Neumann and Parker, 1989; Parker, 1989) was compiled by Neumann and Parker at SRI International Labs. The CERIAS group at Purdue University (Aslam, 1995; Krsul, 1998; Kumar, 1995), contributed towards a computer attack taxonomy, focused on a Unix system. Aslam (1995) and Krsul (1998) extended this work. Kendall (1999) developed a database of computer attacks based on the DARPA intrusion detection evaluation datasets. Lough (2001) proposed a general taxonomy based upon the attack characteristics, but this has shortcomings in practically categorizing attacks like worms, viruses and others. However, Howard at US-CERT (Howard, 1997) outlined one of the most significant taxonomies based on types of attackers, their attack tools, information accessed, attack results and objectives. Atlantic Consulting Services developed the Internet Attack Taxonomy (DeLooze, 2004) in 2000, in which attacks were classified under 4 main categories: Reconnaissance, Denial of Service, Unauthorized Access, and Deception.

There is as yet, no universally accepted taxonomy for computer attacks. One of the reasons for this may be due to the blended nature of attacks. A high coupling may exist between different categories which researchers in the past had difficulty exhibiting with their design. In 2005, Hansman and Hunt (2005) created a taxonomy using four dimensions which addressed: the attack vector, attack target, vulnerabilities and payloads for categorizing computer attacks. They showed the effectiveness of their technique using multiple well-known attacks. This taxonomy can serve as a means to understand attacks and help in labelling them. In order to understand and categorize attacks for labelling purpose, we use features from the taxonomies of Hansman and Hunt (2005) and Lazarevic et al. (2005), some important features that may help in categorizing attacks are illustrated here.

1. Attack Vector or Type

- This includes: Virus, Worms, Buffer Overflow, Denial of Service (DOS), Distributed Denial of Service (DDOS), Remote or Network Attack (including P2P network attack and Wireless network attack), Physical Attack and Reconnaissance or Information gathering attack.

2. Attack Target
3. Vulnerability
4. Attack Outcome
    - This includes: Corruption of Information, Disclosure of Information, Theft of Information, Subversion[20], Remote to Local[21], User to Root[22]
5. Sophistication and Automation: The use of sophisticated tactics and intelligent automated tools to carry out attacks. (Hansman and Hunt, 2005)

2.3 Data Collection for Intrusion Detection

In order to evaluate, train and test intrusion detection systems, it is important to collect suitable labelled data and correlate relevant information and malicious events from it. From the literature review it has been observed that data for intrusion detection is usually collected from sources such as:

1. Networks: data packets
2. Host: Input commands from users
3. Host: System calls, log files, system usage statistics.

Since the focus of our thesis is on NIDS, therefore, we consider network sources. Datasets from network sources may be artificial or synthesized datasets, or non-artificial datasets. Some popular artificial datasets used for performance evaluation measures in the intrusion detection domain include: The DARPA-Lincoln datasets and the KDD99 datasets. Most anomaly detection systems depend on statistical features extracted from the network traffic such as: source and destination ip, source and destination port, latency changes, arrival rates, traffic volume. Many datasets created by researchers were designed with these features in mind. Such datasets lacked payload or content information.

[20]The use of a compromised systems resources for attackers gain

[21] Malicious attacker gains local privileged access over the network or Internet

[22]Attacker is able to escalate/elevate his privileges to administrator or root level (Lazarevic et al., 2005)

The DARPA-Lincoln datasets were synthesized and collected at MIT's Lincoln Labs for IDS performance evaluation. It is available in the form of two datasets known as DARPA 1998 and DARPA 1999 datasets[23], also known as the IDEVAL corpus. Both datasets are in TCPDump and BSM[24] format. The DARPA 1998 dataset contains seven weeks of training data and two weeks of test data. This includes 300 instances of 38 different attacks that can be categorized into four main attack categories, against a Unix host. The DARPA 1999 dataset contains three weeks of training data and two weeks of test data, with around 58 attack types launched against a Unix host, a Windows NT host and a Cisco Router.

The KDD99 dataset was derived form the DARPA 1998 network dataset. This dataset was used in the KDD Cup competition. Features were extracted from TCP connections and labelled as normal or attack traffic. The training set comprises of 24 attacks while the test set contains 38 attacks.

Other artificial datasets can be created employing frameworks to generate malicious traffic. Two notable frameworks include: MACE (Sommers et al., 2004) and FLAME (Brauckhoff et al., 2008).

(McHugh, 2000) critically analysed the DARPA dataset and concluded that the results of artifical or synthetic data are not sufficiently similar to real network traffic data, and hence the DARPA or IDEVAL corpus is not analogous to the properties of real network traffic. This was highlighted with statistics and rate difference between synthetic and original traffic, since the KDD99 dataset was extracted from the DARPA dataset

Real or non-artificial datasets are now being generated to benchmark IDS by researchers. Self-produced datasets promise a larger training set, since malicious streams are manually extracted and labelled from live networks. The procurement of real datasets can be facilitated by the use of Honeypots and Honeynets. In order to get a better insight into computer attacks and create a labelled dataset of such attacks, we employed Honeypots and Honeynets as a data collection tool to evaluate the methods developed in our thesis.

[23]`http://www.ll.mit.edu/IST/ideval/data/data_index.html`

[24]Basic Security Module, logs audit data

2.3.1 Honeypots and Honeynets

Honeypots and Honeynets provide the means to study malicious attacks and attackers under a microscope. Because of their unique architecture they provide a strategic advantage over other security tools by allowing researchers access to critical information regarding attacks as they occur on live systems, at both network and system level. This extra insight helps researchers to quickly devise defence mechanisms to fend off any new or novel attacks. They serve as a means to spy on malicious attackers, their organization, their tools and their sophisticated techniques. Since all traffic received by the Honeynet and or Honeypot is considered malicious, it becomes easy to focus on attack data compared to gateway devices, which cannot provide this much granularity due to the scale or high volume of traffic they process.

2.3.1.1 Honeypots

A Honeypot is generally defined as a network security resource whose value lies in it being scanned, attacked, compromised, controlled and misused by an attacker to achieve his malicious goals. Lance Spitzner defines Honeypots as: "An information system resource whose value lies in unauthorized or illicit use of that resource" (Spitzner, 2002). Honeypots can be classified into two main categories. Firstly, they can be based upon their level of interaction with an attacker. This can be further categorized as:

Low-interaction Honeypot Emulate a variety of host services. These mimic real services but are implemented as a sandbox environment and run as an application. e.g. honeyd and nepenthes. (Provos and Holz, 2007)

High-interaction Honeypot Attacker is given the freedom to interact with a real operating system and their every attempt is logged and accounted for.

Hybrid A hybrid Honeypot would be a mixed type of Honeypot, combining features and functionalities from both low and high interaction Honeypots.

Honeypots can also be categorized by the way they are deployed in a network. This can include server-side Honeypots, which are deployed on a server or host running services, malicious attacker actively attack and exploit these services. Honeypots can also be deployed as client-side Honeypots, which act as clients and actively interact with a malicious server.

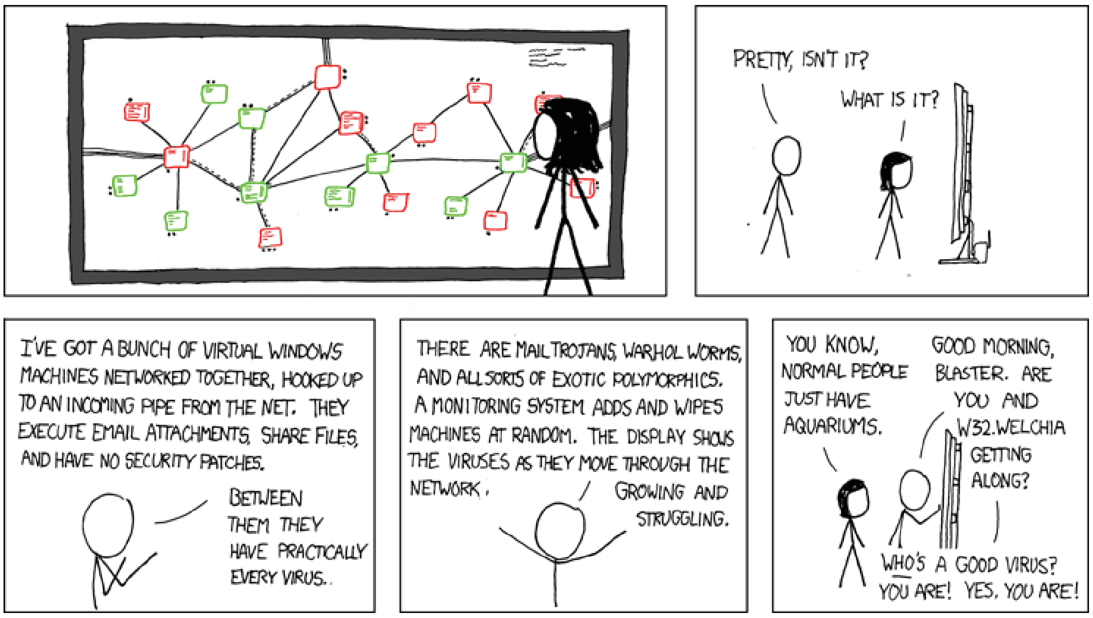

PRETTY, ISN'T IT?
WHAT IS IT?
I'VE GOT A BUNCH OF VIRTUAL WINDOWS MACHINES NETWORKED TOGETHER, HOOKED UP TO AN INCOMING PIPE FROM THE NET. THEY EXECUTE EMAIL ATTACHMENTS, SHARE FILES, AND HAVE NO SECURITY PATCHES.
BETWEEN THEM THEY HAVE PRACTICALLY EVERY VIRUS.
THERE ARE MAILTROJANS, WARHOL WORMS, AND ALL SORTS OF EXOTIC POLYMORPHICS. A MONITORING SYSTEM ADDS AND WIPES MACHINES AT RANDOM. THE DISPLAY SHOWS THE VIRUSES AS THEY MOVE THROUGH THE NETWORK,
GROWING AND STRUGGLING.
YOU KNOW, NORMAL PEOPLE JUST HAVE AQUARIUMS.
GOOD MORNING, BLASTER. ARE YOU AND W32.WELCHIA GETTING ALONG?
WHO'S A GOOD VIRUS? YOU ARE! YES, YOU ARE!

Figure 2.3: A Honeypot comic by XKCD(http://xkcd.com/350/), showing several honeypots connected together in a network, running samples of malware

Honeypots are now widely being deployed and used by security researchers and rolled out in production networks to monitor malicious and suspicious activities. The concept is also illustrated in a light-hearted way in Figure 2.3.

2.3.1.2 Honeynet

A Honeynet is a special kind of high-interaction Honeypot. Honeynets extend the concept of a single Honeypot to a highly controlled network of Honeypots. A Honeynet is a specialized network architecture configured in a way to achieve:

- Data Control: It deals with the containment of activity within the Honeynet.
- Data Capture: It involves the capturing, monitoring and logging of all threats and attacker activities within the Honeynet.
- Data Collection: Captured data is securely forwarded to a centralized data collection point.

This architecture creates a highly controlled network, in which one can control and monitor all kinds of system and network activity. Honeypots are then placed within this network. A basic Honeynet comprises of Honeypots placed behind a transparent gateway the Honeywall. Acting as a transparent gateway the Honeywall is undetectable by attackers and serves its purpose by logging all network activity going in or out of the Honeypots.

Implementation and design challenges of Honeypots and Honeynets relevant to our thesis are discussed in Chapter 3.

2.4 Summary

This chapter provided a brief history, taxonomy and review of the state of the art in intrusion detection technologies, followed by an introduction to computer attacks and a need to study attacks for categorizing and labelling them. This was followed by a discussion on data collection challenges in IDS and a brief introduction of Honeypot and Honeynet technologies as tools for procuring real network datasets for evaluating IDS.

There has been a significant amount of work done on intrusion detection technologies, some of which has been reviewed in this chapter. The literature has demonstrated a clear shift from traditional statistical and pattern matching intrusion detection schemes to hybrid data mining and machine learning approaches. Techniques involving information-theoretic measures have shown promising prospects and have recently emerged as more simple techniques for intrusion detection. For NIDS, it has been observed that most techniques rely on features extracted from packet header data. Such techniques become unreliable and easy to evade (Ptacek, 1998). Researchers are now employing techniques that involve payload analysis.

An understanding of computer attacks can be used to categorize and label attacks to create a training set to train an IDS and later create a larger set to test it. Researchers in the past have used artificial or synthetic datasets for IDS benchmarking, but such datasets have problems associated with them, as they fail to exhibit real network traffic characteristics and may mislead the IDS. There is a need to create datasets based on real or live traffic captures. Honeypots and Honeynets prove to be useful tools to collect and study attacks.

Chapter 3

Architectural Design, Honeynets and Datasets

"Many of the nation's essential and emergency services, as well as our critical infrastructure, rely on the uninterrupted use of the Internet and the communications systems, data, monitoring and control systems that comprise our cyber infrastructure. A cyber attack could be debilitating to our highly inter-depended Critical Infrastructure and Key Resources (CIKR) and ultimately to our economy and national security."
– Homeland Security Council, National Strategy for Homeland Security, 2007

3.1 Problem Description

Security tools provide a line of defence against malicious attacks and attackers. Most of these tools are reactive in nature and attempt to respond only when a known attack has been detected. These security tools, when not properly updated or configured can result in a false sense of security, as attack variants and other evasion techniques can evade these security mechanisms. Attackers are getting more sophisticated and it is becoming difficult to cope with the threats they pose with conventional security tools.

In this chapter we shall discuss:

- The design overview of the proposed IDS.

- The database that is used for evaluation.
- Honeynet architectures and deployment challenges.
- Our implementation of a Virtual Honeynet.

3.2 Design Overview

The general design details of our proposed IDS are described in this section. From a design perspective, our proposed IDS system comprises of the following main components:

1. Malicious Stream Profiler
2. Malicious Stream Processor

These components help us in creating and labelling the ground truth and provide an overview of the various components involved in processing and intrusion detection.

3.2.1 The Malicious Stream Profiler

The malicious stream profiler creates a dataset of known malicious streams. This is done by creating and assigning labels to any given malicious network stream. This serves as a reference (training) dataset of various malicious behaviours. The system is illustrated in Figure 3.1. Details of known malicious streams are studied from sources like CVE, Cert, Snort and samples are collected and extracted from various Honeypot sensors. Unique attack characteristics are identified from these malicious streams. In the absence of real exemplars or samples of malicious streams in the form of PCAPs, packets are crafted or synthesized based on this information containing the actual malicious payload. In order to validate these streams, they are tested with Snort to verify that they are being detected and later sent to the packet profiler module. The packet profiler creates a profile from a stream by stripping the packet headers and extracting the payload. This payload is then written out to raw text files. Labels generated from the validation phase are used to label the malicious stream profile. The profiles of these malicious packets are then added to a set of packet profiles of benign user traffic available for free from `http://www.wireshark.com` and `http://www.pcapr.net` and labelled manually.

Once a mixed dataset of profiles containing both benign and malicious streams is created, they are then sent to the similarity calculator module. This module calculates the similarity between these profiles to yield a pair-wise distance matrix of all packet profiles. The similarity calculation module uses similarity metrics (discussed in the next chapter) for similarity measurements. Each numerical value in the matrix is a representative similarity score between 0 and 1, with 0 being the best match and 1 being the worst match. These similarity scores are then analysed to determine a similarity threshold or cut-off value. This threshold or cut-off value might vary between classes and determines the size of the class. This technique gives us the advantage of being able to accurately estimate the threshold value and avoid any false positives that may arise due to wrong estimation of the threshold. A false positive (FP) in this case would be a benign packet/stream being classified as a malicious packet/stream, due to similarity indicated by the threshold. For all such situations, the threshold value should be lowered enough to avoid any false positives (FP). Once the label and the optimum threshold value is determined, this information is saved inside the labelled malicious stream database. This database becomes our reference malicious network stream signature database. This would be helpful to determine malicious stream activity and behaviour over the network.

3.2.2 Malicious Stream Processor

This system component is in charge of processing network packets or streams to determine malicious behaviour in them. It can detect known malicious streams and their variants and identify novel streams. This is the main detection engine of our proposed IDS. The detection process is similar to the malicious stream profiler with few modifications. The system is illustrated in Figure 3.2. All packets are read from the network device or from a PCAP file and profiled by the packet profiler. As an example, a set of 100 packets or streams are collected and profiled this way. Malicious profiles are exported as signatures or exemplars from the reference malicious streams database and added to the set of 100 captured profiles by the stream adder module. This combined dataset is then sent off to the similarity score calculator module. The similarity score calculator module calculates a pair-wise distance matrix of similarity scores between the collected or incoming profiles and the reference profiles. These scores are then compared with those of the reference malicious streams, and similar packets are clustered together into groups using a clustering algorithm. Any packets or streams found within the threshold range of a reference sample are marked and the classification is updated. These packet profiles display a similarity with the malicious profiles and are thus marked as having malicious activity. This information is inserted into the signature database.

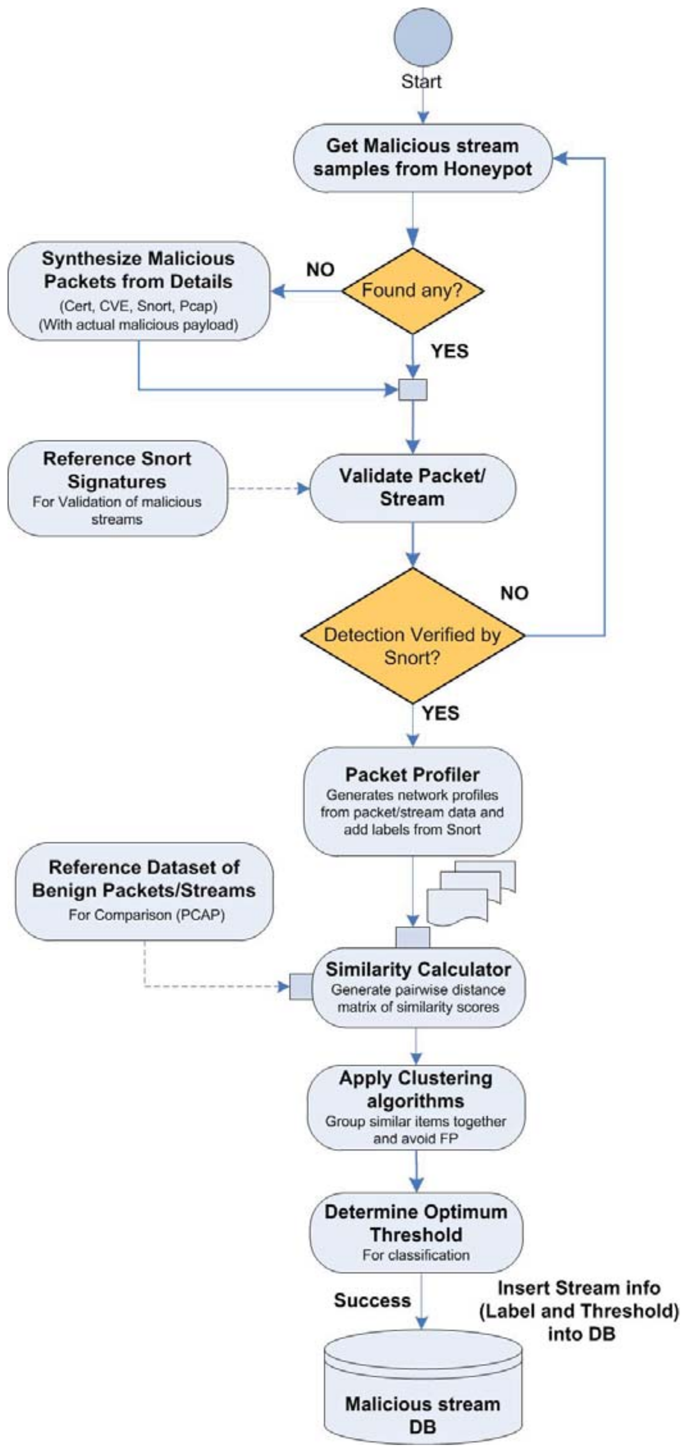

Start
Get Malicious stream samples from Honeypot
Found any?
NO
Synthesize Malicious Packets from Details
(Cert, CVE, Snort, Pcap)
(With actual malicious payload)
YES
Reference Snort Signatures
For Validation of malicious streams
Validate Packet/ Stream
Detection Verified by Snort?
NO
YES
Packet Profiler
Generates network profiles from packet/stream data and add labels from Snort
Reference Dataset of Benign Packets/Streams
For Comparison (PCAP)
Similarity Calculator
Generate pairwise distance matrix of similarity scores
Apply Clustering algorithms
Group similar items together and avoid FP
Determine Optimum Threshold
For classification
Success
Insert Stream info (Label and Threshold) into DB
Malicious stream DB

Figure 3.1: Malicious Stream Profiler: Create and label malicious streams

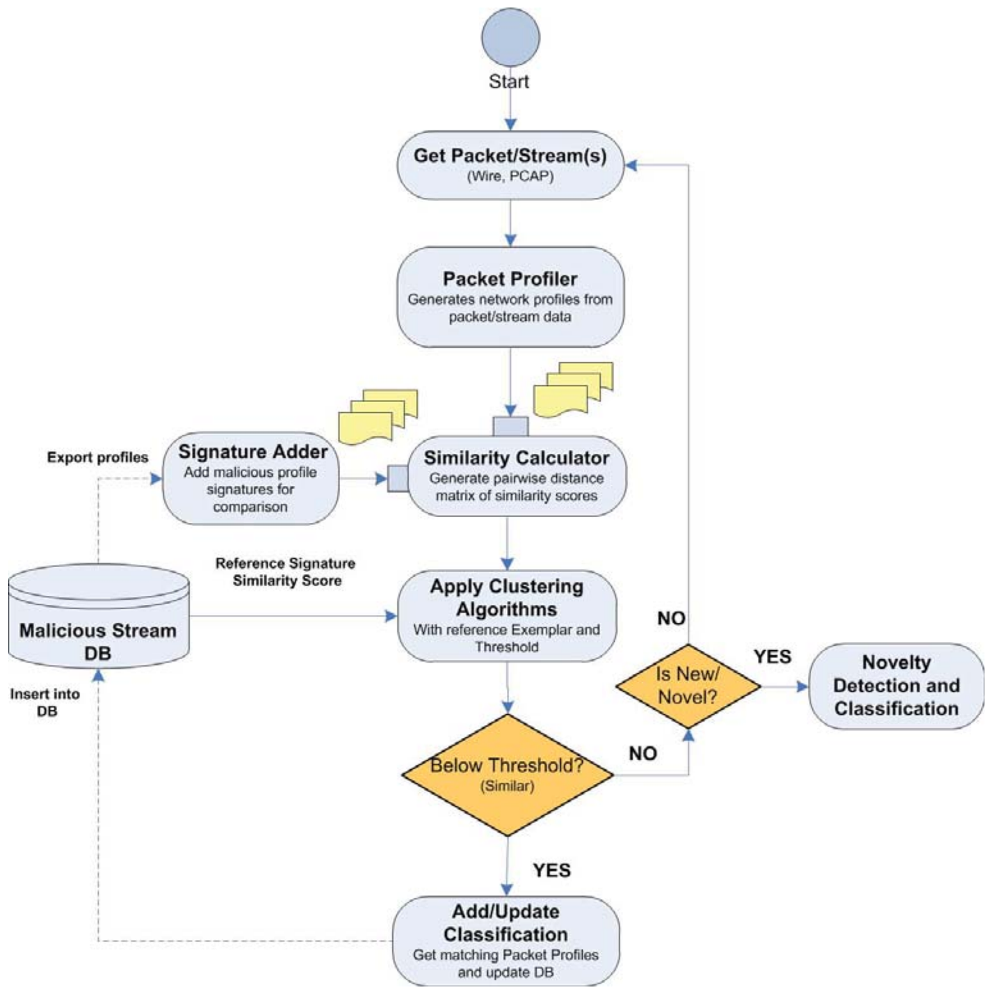

Start
Get Packet/Stream(s)
(Wire, PCAP)
Packet Profiler
Generates network profiles from packet/stream data
Signature Adder
Add malicious profile signatures for comparison
Similarity Calculator
Generate pairwise distance matrix of similarity scores
Export profiles
Reference Signature Similarity Score
Malicious Stream DB
Apply Clustering Algorithms
With reference Exemplar and Threshold
Insert into DB
NO
Is New/ Novel?
YES
Novelty Detection and Classification
Below Threshold? (Similar)
NO
YES
Add/Update Classification
Get matching Packet Profiles and update DB

Figure 3.2: Malicious stream processor: Capture and compare network profiles

3.3 Honeynet Architectures and Deployment Strategies

In the previous chapter we highlighted how Honeypots can be used to collect real data for IDS training and benchmarking. There are several strategies for deploying Honeypots. Most commonly they are deployed as stand-alone systems facing the Internet. Other more advanced strategies involve installing them in a specially crafted network or a Honeynet. Depending on the technologies adopted, and the way data capture, control and collection activities have been carried out within the Honeynet network, the Honeynet has evolved across three architectures or generations over the years, as outlined below:

Generation I or Gen I Honeynet was developed in 1999 by the Honeynet Project. The architecture was simple with a firewall aided by an IDS as the gateway and Honeypots placed behind it. This architecture required two interfaces on the Honeywall gateway, one facing the external network and one facing the Honeypots internal network. This architecture was flawed as the gateway acting as a Layer 3 device could be detected by attackers.

Generation II & III Honeynets brought about a change in the architecture by the introduction of a single device that handles the data control and data capture mechanisms of the Honeynet called the IDS Gateway or the Honeywall. This is implemented as a transparent bridge.

Gen II Honeynets were first introduced in 2001 and Gen III Honeynets were released at the end of 2004. Gen II Honeynets were made in order to address the deficiencies in Gen I Honeynets. Gen II and Gen III Honeynets have the same architecture, with the only difference being improvements in deployment and management in Gen III Honeynets along with the addition of a Sebek server built in the gateway this is known as the Honeywall. This architecture incorporates three interfaces on the Honeywall. Two interfaces acted as a bridge between the external network and the internal Honeypot network; whilst the third interface was used for management and configuration tasks.

3.3.1 Virtual Honeynet

Virtualization technology allows running multiple virtual machines on a single physical machine. Each virtual machine can be an independent Operating System installation, running concurrently with others. This is achieved by sharing the machines physical resources such as CPU, memory, storage and peripherals through specialized software across multiple environments. This reduces project hardware costs. A virtual Honeynet is a complete Honeynet running on a single computer in virtual environment.

VMware Server (Hammersley, 2006) was used as the virtualization solution for our project. VMware Server was selected because it is free, reliable, has large community support, extensive documentation, and the author had previous experience of managing virtual machines with VMware, it is flexible and has robust networking components. For the scope of our project, we used VMware Server version 1.0.6 for Fedora Linux. Later we moved our architecture to a virtualized node running VMware ESX server.

3.3.2 Data capture and analysis tools On the Gateway

For our project implementation we created a prototype using free or Open Source tools. The Honeynet Project[1] is a non-profit, Open Source security research organization. This organization has actively published papers and developed and contributed Open Source security tools. For the scope of our project we used Honeywall CDROM Roo (Project, 2005) version 1.4. Honeywall CDROM is a bootable CDROM operating system built on CentOS for installing, deploying and maintaining a Honeynet. The purpose of the Honeywall CDROM is to automate the installation and maintenance of a Honeynet and provide data analysis support for all activity within the Honeynet. Honeywall Roo includes many well-known security tools such as:

- Snort (Roesch et al., 2006): Intrustion Detection System (IDS)
- Snort_inline: Intrusion Prevention System (IPS).
- Argus (Argus Project, 2008),
- Pof: Passive OS fingerprinting tool
- Tcpdump: Viewing of packet headers.
- Hflow2: A data coalescing tool for Honeynet data analysis.
- Walleye: A web-based interface for Honeywall configuration, administration and data analysis.
- Sebek (Project, 2008): Sebek is a data capture tool designed to stealthily capture attacker's activities

Balas and Viecco (Balas and Viecco, 2005) have given a generalized data collection and fusion diagram for a Generation III Honeywall. Extending their work further, we propose an extended architecture for Honeywall Roo (Project, 2005). Its logical design is shown in Figure 3.3:

3.3.3 Data capture tools on the Honeypot

We implemented a standard Ubuntu server as our Honeypot. This server had basic services running that included SSHD, FTPD and HTTPD etc. OpenSSH was patched and custom compiled to add both a user name and a password logging capability. The passwords were logged to a secure hidden directory within the server. A bash script

[1]`http://www.honeynet.org`

was used to parse the logs and associate them with attacker IPs. Based on a timestamp difference, this script could distinguish between SSH scanners/crackers and interactive SSH sessions. Sebek was also installed on the Honeypot and concealed as a printer driver name. It collected valuable system information such as attacker keystrokes, processes and associated system calls. After analysis of script and Sebek data with Honeynet flows, we could correlate attacks and attack techniques effectively.

3.4 Proposed Honeynet Architecture and Deployment Challenges

The Honeynet Project provides some general documentation on deploying Generation III virtual Honeynets. This documentation was developed by the Pakistan Honeynet Project Chapter. The document was a How-To for deploying virtual Honeynets using VMware. This served as a standard template for anyone wanting to deploy a virtual Honeynet using VMware and Honeywall Roo.

3.4.1 Problem Identification and Proposed Solution

During our literature review it was decided to use the design suggested by Shuja (2008) as the standard template for our project's implementation. Using VMware, a bridged interface like vmnet0 has direct access to the physical network interface. Bridging two such interfaces will cause a bridge between the same LAN segment resulting in loops in the network, whereas the requirement was to bridge between two different LAN segments i.e. the external network segment pointing to the router and the internal network segment on pointing towards the Honeypots. It was observed that the Honeynet design suggested by Shuja (2008) configured both eth0 & eth1 interfaces as a VMware bridge interface and eth2 as a VMware host-only interface. This was causing loops in the topology and the Honeypot LAN segment was being avoided and unroutable. Similar design problems were being faced and discussed by security researchers all over the globe who wanted to implement a similar virtual Honeynet. Many of these questions appeared in the official Honeywall community mailing list. I communicated this problem and proposed solution to the Pakistan Honeynet Project with a complete proof of concept in June, 2008. They accepted and updated the design on their website.

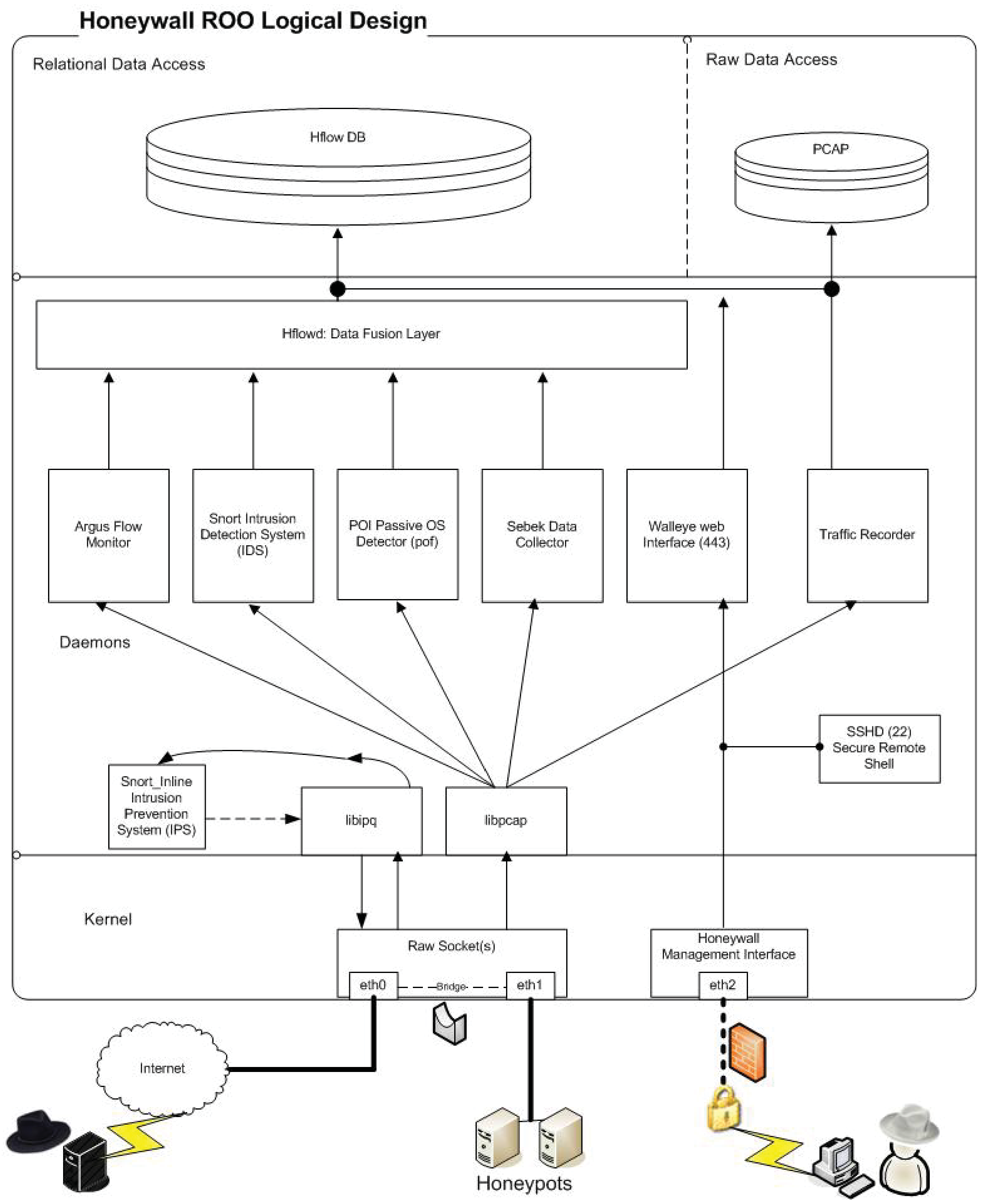

Honeywall ROO Logical Design
Relational Data Access
Raw Data Access
Hflow DB
PCAP
Hflowd: Data Fusion Layer
Argus Flow Monitor
Snort Intrusion Detection System (IDS)
POI Passive OS Detector (pof)
Sebek Data Collector
Walleye web Interface (443)
Traffic Recorder
Daemons
SSHD (22) Secure Remote Shell
Snort_Inline Intrusion Prevention System (IPS)
libipq
libpcap
Kernel
Raw Socket(s)
eth0
Bridge
eth1
Honeywall Management Interface
eth2
Internet
Honeypots

Figure 3.3: Honeywall Roo Logical Design

3.4.2 Design Details and Discussion

We shared and discussed our findings with the community on the Honeywall project mailing list. After necessary testing, a design for a data collection Honeypot was prepared and followed for the project implementation. Based on the success of this effort it was decided to publish the improved design. This design proposes 3 virtual interfaces for the virtual Honeywall such that:

1. Interface vmnet0 (on eth0) is a VMware bridge interface pointing towards the router. (As shown in Figure 3.4)

2. Interface vmnet1 (on eth1) is a VMware host-only interface leading to the internal LAN segment where Honeypot(s) are kept. (Also shown in the figure 3.4).

3. Interface vmnet2 (on eth2) is a VMware bridge interface that is firewalled and accessible for remote management purpose (remote access via SSH and Walleye.)

Interfaces 1 and 2 are picked up by Roo as eth0 and eth1 and are used for bridging. Interface 3 is used for remote management. As shown in Figure 3.4 the doted boxes indicate the publically assigned IP addresses. In this case, the host machine's eth0 interface and the Honeypot virtual machines (1, 2 or many) are assigned public IP addresses. The Honeywall management interface (i.e. interface 3) is assigned a private IP from the Host machines virtual remote management interface, and can be configured in many ways:

1. The interface can be routed to an internal virtual subnet (private IP subnet) on the same physical machine.

2. The interface can be routed to an internal/external routable network using a separate physical Ethernet controller on the physical machine.

3. The interface can be routed on the same subnet as of the Honeynet sharing the same physical machines Ethernet controller.

For our implementation we assigned the management interface a public IP from within the Honeynet subnet, but restricted access to it only from a specific IP (via Roo (Project, 2005) configuration). This project was implemented successfully with one physical gigabit Ethernet interface. Another physical interface could have been used by binding it with the remote management interface

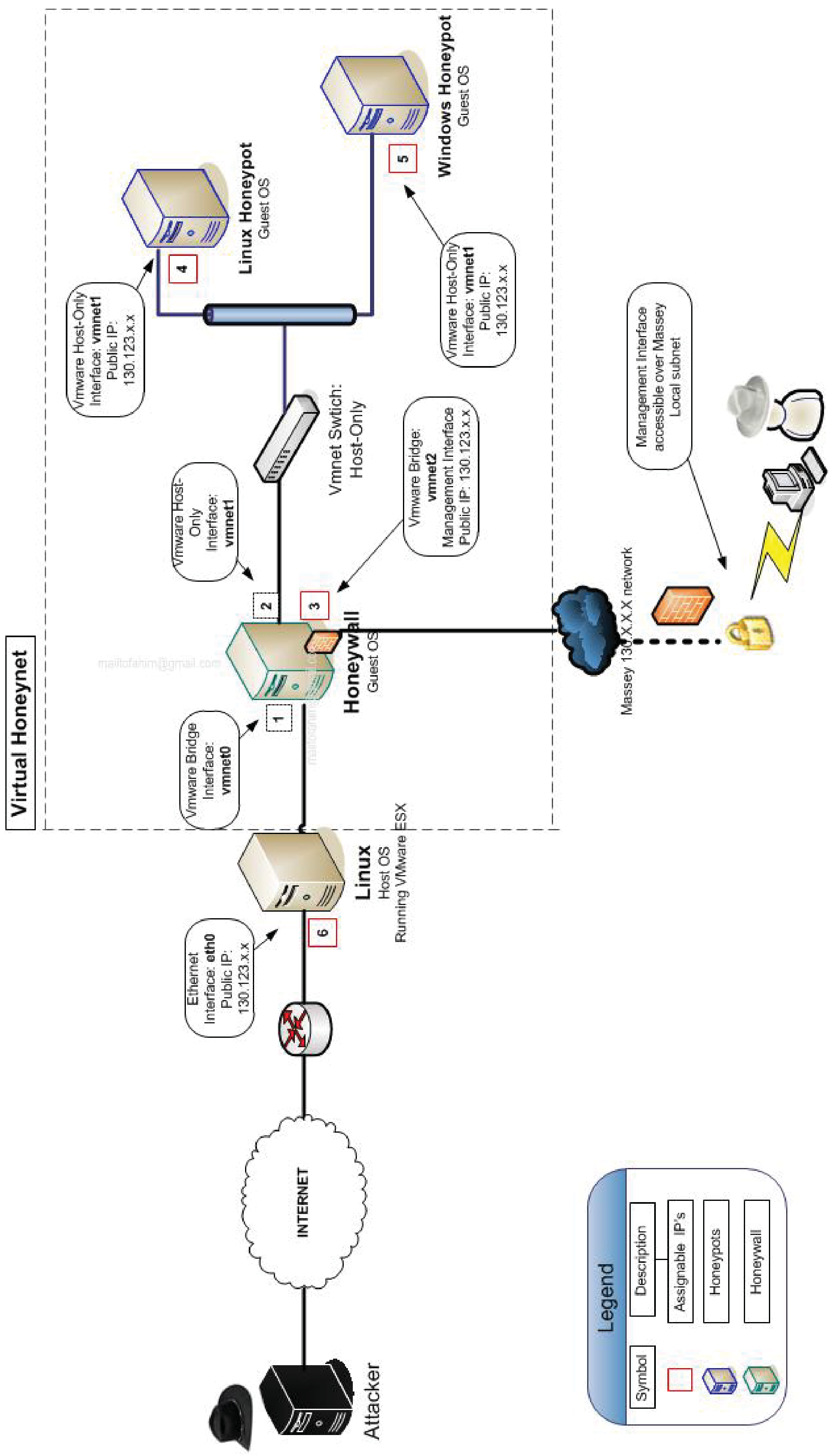

Virtual Honeynet
Attacker
INTERNET
Ethernet Interface: eth0 Public IP: 130.123.x.x
6
Linux Host OS Running VMware ESX
Vmware Bridge Interface: vmnet0
1
Honeywall Guest OS
2
3
Vmware Host-Only Interface: vmnet1
Vmware Bridge: vmnet2 Management Interface Public IP: 130.123.x.x
Vmnet Swtich: Host-Only
Vmware Host-Only Interface: vmnet1 Public IP: 130.123.x.x
4
Linux Honeypot Guest OS
5
Windows Honeypot Guest OS
Vmware Host-Only Interface: vmnet1 Public IP: 130.123.x.x
Massey 130.X.X.X network
Management Interface accessible over Massey Local subnet
Legend
Symbol
Description
Assignable IP's
Honeypots
Honeywall

Figure 3.4: Virtual Honeynet Proposed Design

3.4.3 Results and Discussion

The virtual Honeynet was online for a period of approximately 60 days from 15th September 2008 to 15th November, 2008. During this period we received approximately 30,000 identifiable attack connections. The attack results were documented as attacked ports and services, Attacker IPs and Country of Origin. The first attack was documented after 4 days of setting up the Honeynet. After several port scans an attacker attempted a SSH brute force attack from 82.99.xx.xxx. Geo-location of the IP was retrieved (Padmanabhan and Subramanian, 2001). After several hundred attempts the attacker was successful in brute forcing a user account. A botnet client was installed from a free webhosting server and IRC (Oikarinen and Reed, 1993) communication was initiated, the chat sessions were translated from Romanian to English using Google Translate service. The tools and chat/commands were retrieved from this session successfully for further forensic analysis. During the project, five similar sophisticated attacks were observed, from which valuable information and tools have been successfully retrieved. Forensic analysis have revealed a depth of information on the attackers, their organization into groups, their ties with each other and some system credentials were logged during the chat exchange. These forensic results are outside the scope of this thesis. However, we came to the conclusion that attackers originating from Europe are commanding a large army of zombie hosts in China and the US to gain access to targets across the globe. Servers are always a highly valued target for them as they offer a variety of services over stable high speed links. The malicious streams collected using our virtual Honeynet setup have been used to train and test our proposed IDS. Some statistics on attacks are detailed in the following sub-sections.

Attacked Ports and Services

A small observation of attacked ports and services was made, which revealed that, out of a total of 29,643 probed ports and services, 29,048 were targeted at SSH. This indicates the attackers' focus on brute force means of gaining access to the server. This is followed by high activity on IRC ports indicating botnet activity.

Attacker IPs

During its 60 day tenure, the Honeypot received 34263 attacks from 615 unique IP's. A great many of these attacks originated from Europe and China, followed by the US, as shown in Figure 3.5 and 3.6

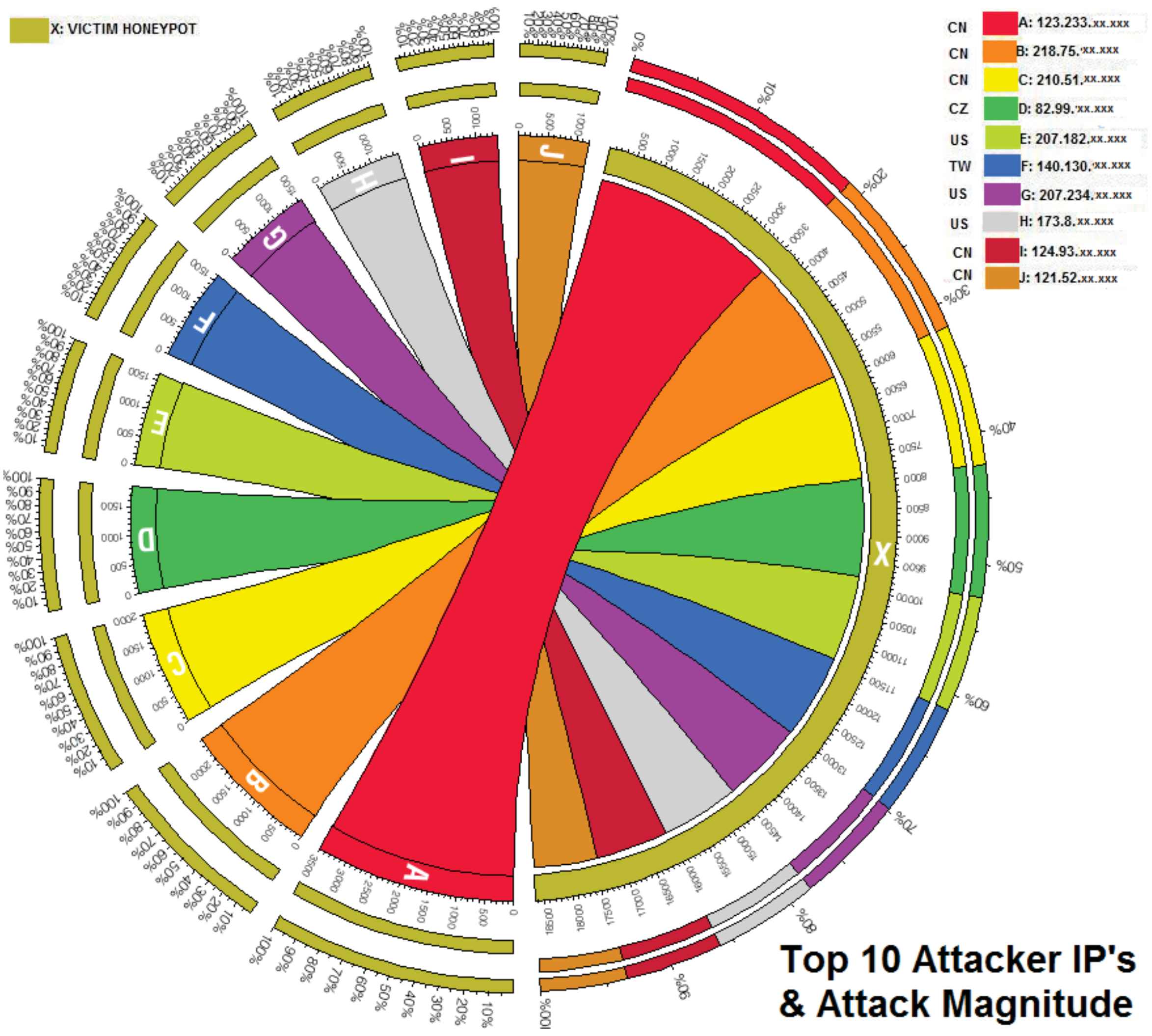


Figure 3.5: Top 50 Attackers by IP Address

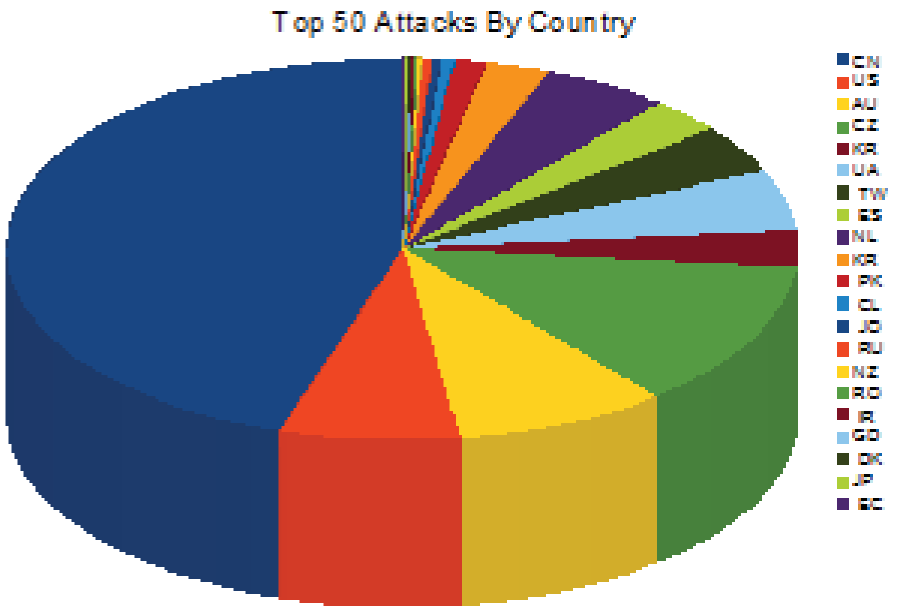


Figure 3.6: Top 50 Attackers by Country

Attackers Country of Origin

615 unique attacker IP addresses were identified originating from 79 countries across the globe. Out of these 79 countries the highest number of attacks came from China and Europe followed by the US. The same proportion also stands for the highest attack frequencies as shown in Figure 3.5 and 3.6.

3.4.4 Honeynet Architecture Summary

The hardware and software used for this project has been summarized in Table 3.1. It can be inferred that standard hardware can easily be used to setup a virtual Honeynet. Large amounts of memory are always a preference for virtualized environments and can be used as a performance benchmark. The availability of free and Open Source tools and technologies such as Linux, VMware, Snort, Argus and those provided by the Honeynet Project have considerably eased the process of setting up such projects. However, such tools demand a high degree of skill and customization, along with a thorough understanding of the system.

We successfully setup and maintained a Generation III Virtual Honeynet using free and Open Source software, along with standard hardware. This led to a very nominal implementation cost for the project. A great deal was learned from this experience

Project Summary		
Feature	**Product**	**Specifications**
Host Operating System	Linux, Fedora 9	HW Vendor: Dell Optiplex 755, Processor: Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz, L2 Cache: 4MB, RAM: 4GB , Storage: 250GB, NIC: 1GB Ethernet controller (public IP)
Guest Operating System 1 (HONEYWALL)	Linux, Honeywall Roo 1.3	Single Processor Virtual Machine, RAM: 512 MB, Storage: 100 GB, NIC 1: 100Mbps Bridged interface vmnet0, NIC 2: 100Mbps host-only interface vmnet1 , NIC 3: 100Mbps Bridged interface vmnet2 (public IP)
Guest Operating System 2 (HONEYPOT)	Linux, Ubuntu 8.04 LTS (Hardy Heron)	Single Processor Virtual Machine, RAM: 256 MB, Storage: 20 GB, NIC: 100Mbps host-only vmnet (public IP)
Virtualization software	VMware Server	VMware Server 1.0.6 for Linux x86
Architecture	Gen III	Gen III implemented as a virtual Honeynet
Honeywall	Roo	Roo 1.3
IDS	Snort	Snort 2.6.x
IPS	Snort_inline	Snort_inline 2.6.1.5
Data Capture Tool	Sebek	Sebek 3.2.0
Honeynet Project Online Tenure		September 12, 2008 TO November 1, 2008

Table 3.1: Project Summary: November, 2008

and certain areas for improvement have been identified. We have achieved a significant level of familiarity with all the tools utilized for the project and have identified some areas for further tool development and enhancement.

The project was initially setup at Victoria University of Wellington, many thanks to Dr. Peter Komisarczuk and Dr. Ian Welch, who kindly provided the means and equipment to host the project at Victoria University of Wellington. Since then we have setup a virtual honeynet node at Massey University Campus, running special virtualized hardware managed by the VMware ESX software. We are now running multiple honeypots including:

- Dionaea: for stream payload and malware collection.
- Kippo: as a SSH Honeypot.
- Glastopf: as a webserver honeypot.
- Cuckoo box: as a sandbox for malware analysis.

- Several high-interaction windows and linux honeypots (real operating system with real services).

For this thesis, data from the Dionaea sensor was used for analysis. Other Honeypot sensors listed here do contribute data, but are currently not used in the analysis.

From our experience with Honeynets it can be concluded that, although Honeynets are excellent tools for data capture, there is still room to enhance and automate them. The current mechanism lacks correlation of network and host events. Automated attacker profiling can also be implemented in Sebek. It is believed that virtual Honeynets can serve as an excellent environment for collection of malicious streams and to aid detection of existing IDS. Finally, the Hflow database may serve as an excellent platform for building feature rich future security applications.

3.5 Dataset

To evaluate the effectiveness of an IDS or to compare its performance, we require datasets of malicious traffic, or attack generation tools. In the absence of standard IDS evaluation data sets, it is difficult to establish an unbiased evaluation of its detection capability. A good test dataset should contain a greater coverage of the various types of network traffic including labelled attacks, while also accounting for the properties and nature of network traffic. Unfortunately no such standard dataset could be found.

Various research groups have maintained their own datasets for evaluation, but they aren't very reliable. Very few datasets are publicly available, many of which contain only features of statistical importance extracted from packets. Many datasets used by researchers focus on statistical features extracted from network traffic and mainly headers to identify anomalies instead of actual network dumps. Features such as source and destination IP and ports, packet size, header size, bytes sent and received, flags, protocol type fields and connection rate. These features may serve to identify some types of anomalous traffic, but cannot serve as a general rule for sophisticated attacks. We are more interested in the packet/stream payloads, as they contain representative digital fingerprints of exploits. The closest to a good (and to some extent a standard) dataset is the MIT DARPA 1998 and 1999 dataset also known as the IDEVAL corpus. Though dated, the dataset comprises of categorized or labelled traces of benign as well as malicious network activity. McHugh (2000) gave a critical analysis and determined that results for synthetic data are not practical to real data. Hence it is concluded that the IDEVAL corpus is not analogous to the properties of real network traffic. Exploit frameworks like Metasploit[2] could also be used to create a labelled dataset for IDS testing, however real attacks seen by honeypots usually have variations from known attacks and provide a chance to detect novel attacks.

Tillmann Werner employed honeytrap[3] honeypots to create a manually analyzed and labelled dataset for Nebula (Werner et al., 2009). He was kind enough to share this dataset for our IDS evaluation. Honeytrap is a low interaction honeypot that is capable to capture all network streams directed towards it. This dataset comprises of a set of 6631 unique malicious streams and benign, but suspicious streams collected from two honeytrap honeypot instances during the last quarter of 2007.

These malicious and benign streams comprising of categories including: TFTP download, Buffer Overflow (LSASS), Shellcode, TFTP download, Oracle DB Connection Attempt, Remote Framebuffer Protocol (VNC) Version String 005, Big Yellow (Symantec) Buffer Overflow Exploit, TrendMicro ServerProtect Buffer Overflow Exploit, FTP download (via cmd.exe), Webmin Exploit GET /unauthenticated/..%01/..,RPC/DCOM Buffer Overflow Exploit with simple polymorphic random data, ASN1 Buffer Overflow Exploit (Shellcode 1), MySQL UDF Injection and Execution with FTP download, MySQL root access, SMB/CIFS Scan, HTTP GET Request (absolute http Path), HTTP GET Request /w00tw00t.at. ISC.SANS.DFind:), HTTP OPTIONS Request with User Agent Infos, POP3 Login Attempt, SIP scanner, ack, invite, cancel, MS-directory service (port445), NMAP SERVICE PROBE, MMS stream, EPMAPPER (port 135) and PROPFIND WEBDAV. The system is also tested on this dataset and results are discussed in Chapter 5 and 6. This dataset contains 28 main labelled categories and a total of 55 labels including all sub categories. The labels for two subcategories a and b of a main category c are expressed by c.a and c.b. The malicious streams range from buffer-overflows, to exploits, to shellcode to worm and worm variants. The benign but suspicious streams include traffic like HTTP GET, POST and OPTIONS requests, FTP and TFTP download requests, Oracle and Mysql database connection requests.

We also created our own dataset using a Dionaea Honeypot. Dionaea (2012) is a low interaction Honeypot that has proven to be quite useful for collecting binary malware samples, binary network streams or payload, shellcode and extensive attack detail. For our implementation we used the latest version 0.1.0. This gives it a clear advantage over honeytrap. It too like honeytrap can detect and log all network streams directed towards it. We manually collected, labelled and classified 6600+ samples using Dionaea bi-streams, and labelled the dataset using Tillmann's methodology as discussed above.

[2]http://www.metasploit.com/
[3]http://honeytrap.carnivore.it/

3.6 Summary

This chapter outlined the design of our proposed IDS and describes the different functional components of this system that will be discussed in detail in the subsequent chapters. It presented a discussion on why Honeypots and Honeynets are important for security researchers and what strategic advantages they provide over other security tools because of their unique design. At the end we discussed implementation challenges and details and finally presented some results, based on our experiences. The chapter concluded with a discussion on the Datasets that are used for evaluation of the IDS.

Chapter 4

Similarity Metrics

"If you cannot measure it you cannot control it."
John Grebe (1990)

In this chapter we shall discuss how distance metrics can be used to determine the similarity between malicious network streams. Similarity metrics are listed and described. The problems and challenges in their use and evaluation are identified. A methodology is developed in response to these challenges, followed by a series of experiments and results. Finally, validation and comparison strategies are investigated to determine the best metric for comparison.

This chapter serves as a foundation for the thesis, in which the similarity measurement techniques are identified. The next chapter discusses the application of clustering and classification techniques on these measures and the chapter after that (Exemplar Learning) discusses life-long learning strategies from an implementation perspective, highlighting algorithms for supervised and unsupervised learning for intrusion detection.

4.1 Problem Description

The problems that we shall address in this chapter include:

1. What metrics are available to measure similarity between similar types of malicious network traffic or streams?
2. What metrics provide the most appropriate methods for making similarity comparisons for our domain?

3. How do these metrics compare?

Signature or Misuse based intrusion detection systems (IDS) use signature or pattern matching techniques to detect intrusive events in packets or streams. A signature is a unique pattern or characteristic used for identification and it is used to "describe the characteristic elements of an attack" (Kreibich and Crowcroft, 2004). It is assumed that network attacks comprise of specially crafted string sequences, used to exploit services and shellcode instructions to execute code remotely on the victim host. Signature matching techniques identify attacks by comparing the contents or byte sequence of packets with a set of signatures or rules that the system has in its signature database. These techniques can become unreliable against evasion techniques like transposition or scrambling of information within a packet or stream (Ptacek, 1998). Variation or permutation of network attacks and exploits may cause problems, as they are undetected by an IDS because the pattern matching mechanism fails to identify the mutated pattern or sequence in that attack variant. Hence, minor tweaks currently slip past automated defences, as they are very order sensitive and modification may change the ordering of byte sequences. Most intrusion detection techniques focus on creating signatures from attack traffic. A large number of false positives (FP) occur when such systems misclassify benign traffic as attack traffic, thus greatly undermining confidence in the reliability of such systems.

Since both forms of variation i.e. transposition or reordering or permutations of symbols and substitution or substitution of symbols cause evasions, both are considered a challenge for IDS. Substitution may change the byte sequence substantially, making it difficult to detect such sequences. Hence, the focus of this thesis will be to detect variants caused by transposition methods or those variants that may inherit symbols or a block of symbols from a common reference i.e they share some analogous information with a reference. This is evaluated experimentally throughout this chapter using a simple, contrived examples.

It is hypothesized that similar types of network traffic can be grouped together based on their entropy or the similarity of characters or byte sequences within them, independent of their ordering. If this similarity X between network traffic profiles can be measured, then it can be reflected as a similarity score. The problem is how to measure and determine this similarity X with confidence.

4.2 Methodology

This section investigates what information in malicious packets and streams can be used as features for similarity measurements.

A significant amount of information can be extracted from any given traffic stream. This information can be processed and filtered for detecting attack variants, prior to sending them to the signature generation sub-system. This can be done with the help of systematic measures that can determine similarity between variants of attack or malicious streams with statistical confidence. The notion that information is quantifiable is well established (Shannon, 2001). However the approach involving measurement of analogous information content in packets using string metrics and semi-metrics, to determine a level of similarity between them is novel. In this section, we shall discuss the various components of our system that enhance an existing signature-based system with the additional capability of detecting attack variations or variations of malicious network streams.

4.2.1 Feature Extraction: Packet/Stream Profiles

For network based measurements and simulations, researchers have utilized 5-tuples [Source IP, Destination IP, Src Port, Dst Port, Protocol] as the basis for detection and analysis of network traffic. This method might be effective in certain problem domains, however, information contained within this tuple can have high entropy due to the changing IP addresses and port numbers targeted during an attack, and thus the IDS fails to perform intrusion detection properly. Furthermore, the defining characteristic of an attack or malicious stream lies in the packet payload. This payload can be used to create a raw profile of a malicious packet or stream. This information can easily be extracted from a flow. A “Flow” is a unidirectional component of a TCP connection (or its UDP or ICMP equivalent). A bi-directional flow maintained by a session (like a TCP session), is called a network stream. From these flows we intend to:

- Extract meaningful features associated with each flow (or group of flows/streams), and
- Use these feature values to determine whether the flow is anomalous or not, by grouping them together based on the similarity between them.

A Packet Profiler subsystem (discussed in Chapter 3: 3) extracts payload information from a packet and writes it into a raw text file to form a behavioural profile. This file is the representative packet profile and will be referred to as a “Packet Profile” in this text. Using similarity metrics we can determine the similarity of information content amongst these various packet profiles. This will help to determine and classify similar packets into groups. Thus, by creating profiles for malicious network traffic, we should be able to identify malicious packets/streams and their variations from network flows.

4.2.2 Measuring similarity between packet/stream profiles?

A Signature 'sig' in an intrusion detection system consists of a set of distinct syntactic features f_i observed from attacks as they occur or have occurred and rules r or actions to take when such a set of features is detected, thus $sig = \{f_i, r\}$. These features f_n form a distinct set of patterns P such that $P := \{f_1, f_2, ..., f_n\}$. Where each individual feature f_i comprises of one or many unique symbols s such that $f_i = s^+$ or bytes b such that $b = (\{0,1\}^8)^+$. Here, P is a set of sub-strings that are unique to an attack. In the case of permuted or scrambled information $I_{scrambled}$ the order of symbols s forming the pattern P changes, but the inherent symbols remain the same. These syntactically distinct but symbolically similar strings can be measured effectively for similarity due to similar corresponding information or symbols. Thus scrambling the information, using transposition ciphers $T(I) = I_{scrambled}$, has little effect on the entropy of the strings. However, if new symbols are added or replaced or substituted, using substitution ciphers $S(I) = I_{substituted}$, the entropy may change drastically.

Based on the above discussion and literature review, it is concluded that two groups of measurement techniques can be used for our domain:

1. Those derived from Mathematics: edit distance based measures.
2. Those derived from information theory: information-theoretic based measures.

Information theory suggests some techniques to determine the similarity between two strings. One way to do this is by using the notion that each string contains information and the amount of information stored in a string is quantifiable and articulated by its Entropy and Kolmogorov Complexity. Another technique involves exploiting the redundancy of information within these strings to determine similarity.

From mathematics, string metrics can be used to determine the approximate similarity or dissimilarity between strings. Applying the same theme to network streams, we can now perform similarity measures of known attacks or malware streams with features extracted from incoming network streams. The advantage over plain misuse based IDS is that instead of matching byte sequences occurring at a certain position in the packet, information patterns within them are matched.

4.3 Similarity Metrics

This section discusses and compares the various methodologies involved in finding the similarity between network profiles. A theoretical overview with some contrived examples is illustrated, followed by a discussion on the criteria for choosing that methodology, compared to the others. This shall be followed by detailed experiments and results to provide empirical evidence to choose the best method in the subsequent sections.

In the previous section it was concluded that since our problem domain involves network or packet profiles expressed as strings or symbols, the metrics can be categorized by their measurement techniques. For this section the following groups will be considered, with several metrics selected from the literature:

1. String Metric (Edit distance based measures)
   (a) Levenshtein Distance
   (b) Demaru-Levenshtein Distance
   (c) Hamming Distance
   (d) Jaro Distance
   (e) Jaro-Winkler Distance.
2. Information-theoretic based measures
   (a) Spamsum
   (b) NCD
3. Hybrid

4.3.1 String Metric or Edit distance based measures

These methods involves the use of string metrics on raw packet/stream profiles for establishing a similarity score for similarity measurements.

Edit Distance Metrics

From the literature, five edit distance based similarity metrics have been short listed. These similarity metrics are used to identify similarities between various network profiles in our dataset. These metrics were selected on the criteria of being most suitable for our domain, well-known, established, and highly cited, which involves string operation between instances of network streams. These metrics are: Levenshtein Distance

(Navarro, 2001), Demaru-Levenshtein Distance (Navarro, 2001), Hamming Distance (Navarro, 2001), Jaro Distance (Cohen et al., 2003), and Jaro Distance (Cohen et al., 2003)

From a classification point of view, these metrics represent a few notable classes. Edit distance is one such class of distance functions, which maps a pair of strings x and y to a real number $\Re$. A smaller value of indicates greater similarity and a larger value represents dissimilarity between x and y. The similarity scores between x and y can be expressed in a vector form such as: x,y,$score$.

Levenshtein distance can be explained as " the minimal number of insertions, deletions and substitutions required to make two strings equal" Navarro (2001). It counts an operational cost of 1 for every insertion, deletion and substitution. It is also known as "string matching with k differences" Navarro (2001). The distance is symmetric, such that:

$$0 \leq d(x, y) \leq \max(|x|, |y|)$$

For any two given strings A and B having i characters. Let a be a character from string A and b represents a character from string B, then:

$Sub(a, b) = 0$, when $a = b$;

$Sub(a, b) = 1$, when $a \neq b$;

$Del(x) = Ins(x) = 1$, when $a \neq b$;

Then $L[m-1, n-1]$ represents the Levenshtein distance between A and B (Levenshtein, 1966).

In contrast to the Levenshtein distance which performs three edit operations (insertions, deletions and substitutions), the DamerauLevenshtein edit distance allows multiple edit operations with transposition as an additional edit operation.

Hamming Distance allows only substitutions, with a cost of 1. It is also termed as "string matching with k mismatches." (Navarro, 2001) The distance is symmetric, such that: $0 \leq d(x, y) \leq |x|$

Jaro Distance and its variants are used often for record-linkage. This heuristic based similarity metric is not based on an edit distance model (Cohen et al., 2003). To determine the Jaro distance between any two given strings x and y, where $x = a_1 ... a_K$ and $y = b_1 ... b_L$, We have, $b_j = a_i$, when any character a_i in x is common with character b_j in y

A transposition $T_{x',y'}$ for a position i such that $x'_i \neq y'_i$ will be when $x' = a'_1...a'_K$ characters in x would be common with $y' = b'_1...b'_L$ in y, The Jaro similarity metric for x and y is

$$Jaro(x, y) = \frac{1}{3} \cdot \left(\frac{|x'|}{|x|} + \frac{|y'|}{|y|} + \frac{|x'| - T_{x',y'}}{|x'|} \right) \tag{4.1}$$

Winkler (Cohen et al., 2003) proposed a variant of the Jaro distance that uses the length L of the longest common prefix of x and y. Given here:

$$Jaro\text{-}Winkler(x, y) = Jaro(x, y) + \frac{L'}{10} \cdot (1 - Jaro(x, y)) \tag{4.2}$$

Comparisons

Test 1 We performed an evaluation of these string metrics with the help of the following basic test. For example, consider two strings 'strA' and 'strB':

*strA = "aaabbbcccdddeeefffggghhhiiijjjkkklll
mmmnnnooopppqqqrrrsssttttuuuvvvwwwxxxyyyzzz"*

*strB = "zzzyyyxxxwwwvvvuuutttsssrrrqqqpppooonnnmmm
lllkkkjjjiiihhhgggfffeeedddcccbbbaaa"*

Here string b 'strB' is the reverse of string a 'strA'. Now we will perform string measurements on these two strings and some variants to show their effectiveness in detecting network attacks:

```
## HAMMING DISTANCE
>>> hamming_distance(strA,strB)
78 or 0.78 on a scale of 0-1
## LEVENSHTEIN AND DAMERAU
>>> levenshtein_distance(strA,strB)
78 or 0.78 on a scale of 0-1
>>> damerau_levenshtein_distance(strA,strB)
77 or 0.77 on a scale of 0-1
## JARO AND WINKLER
```

```
>>> jaro_distance(strA,strB)
0.5085470085470085
>>> jaro_winkler(strA,strB)
0.5085470085470085
```

Here the results are scaled from 0 to 1, where a result of 0 means identical and 1 means highly dissimilar. Here it is important to note that the scaling does incorporate 100 which is scaled down to 1 as the upper limit. For Levenshtein and Hamming distances there could be situations where the upper limit may go beyond 100 (or 1). This could be mainly due to extra or more dissimilarities in unequal sized strings that get accounted for by their respective algorithms. However, for our calculations the upper limit is restricted to 100 or 1, as this is interpreted as 100% dissimilarity and any percentage of dissimilarity beyond that is not required or considered by our proposed system. From the above similarity results its evident that the Jaro and Winkler distance gives the lowest score or greatest similarity between the two strings.

Test 2 In order to determine how sensitive the Jaro metric really is to variants, we perform some additional tests. We now test the Jaro metric for unequal length string comparisons, followed by mismatched strings and their variants:

```
>>> jaro_distance(strA,strB)
0.5085470085470085
>>> jaro_distance(strA,"fahim")
0.4213675213675214
>>> jaro_distance("uvwxyz","abcdef")
1.0
>>> jaro_distance("uvwxyz","abcdefz")
0.6365079365079365
```

The above tests have pointed out a limitation of the Jaro metric, which may cause problems and unreliable results for our domain. The comparison between 'strA' and the string 'fahim' yields a very high score of 0.42, approximately 60% similarity, which is unusual or unexpected for the two strings. For the second and third strings, though strings 'uvwxyz' and 'abcdef' do not share any common characters and therefore are correctly termed as having 0% similarity, with a score or distance of 1, a simple addition of the character 'z' makes them 40% similar, as shown in the next string comparison between "uvwxyz" and "abcdefz". This bias in the results due to a single matching character can lead to ambiguous or unreliable results. Hence it can be concluded that although the Jaro metric can be used to give a good approximation for similarity between similar length strings, its ability to work on unequal length strings having some common characters becomes highly biased. This is discussed in more detail later in the results section 4.5.

Conclusion

From the above experiments and disucssion it can be concluded that Jaro and Jaro-winkler distances provide the best approximation amongst other edit distance based measures tested for our domain. However, Jaro distance does have some sensitivity issues and might yield ambiguous results for partially similar strings.

4.3.2 Information-theoretic measures

4.3.2.1 Information Theory and Security

Information theory states that "Information is a measure" and is based on probability theory and statistics. A key measure of information is entropy, expressed by the average number of bits required for communicating that information. A bit is the unit of information, based on binary logarithm Shannons definition of self-information I(E) of an event E is given by (Shannon, 2001):

$$I(E) = \log_2 \left(\frac{1}{P(E)} \right) = -\log_2 P(E) \tag{4.3}$$

Information theory thus becomes the foundation and motivation for finding information patterns instead of string patterns for intrusion detection. The Entropy of a random variable X is given by:

$$H(X) = -\sum p(x) \log_2 p(x) \tag{4.4}$$

Here X is a set of all messages $x_1, x_2 ... x_n$ and $p(x)$ is probability of X given $x \epsilon X$. Entropy is at its maximum when messages in the ensemble are equi-probable, viz

$p(x) = 1/n$ for all x i.e. most unpredictable such that $H(X) = log_2 n$. Entropy is now widely used for security in areas like the determination of the quality of randomness in a cryptographic system, detection of encrypted and packed malware, the finding of cryptographic material, forensics and network anomaly detection. Each malicious stream carries information in its payload. Even if the attacker attempts to mask this information, it will have a distinct fingerprint. We attempt to measure this information when it is in its deterministic or raw state.

4.3.2.2 Similarity using Spamsum

Spamsum is a program for spam email detection, based on the Context Triggered Piecewise Hash (CTPH) algorithm. Spamsum was developed by Andrew Tridgell (Tridgell, 2002) as a means to detect emails similar to reference spam emails. The concept of Spamsum is suitable for our domain and the idea to utilize it as a measure to detect similar malicious network streams is novel and unique to this thesis. A series of tests are performed to evaluate the effectiveness of this measure for our domain.

Test 1 This methodology involves the use of string metric algorithms (discussed in previous section), on network profiles containing Spamsum hashes.

Scaling down the packet payload with Fuzzy hashing (Spamsum) Ideally, our system should be capable to detect attack variations by observing network packets or streams. Since packet payloads can vary drastically in both length and content during a communication, adding the entire ASCII or Hex payload to the profile can yield abnormal results when calculating similarity. One approach is to create a fingerprint of the entire payload or parts of it that are a unique representation of the payload and yet reproducible, to find similarities when compared with other flows.

A possible solution is to represent variable size payloads as fixed size fingerprints and hash them. A hash is a mathematical process that can generate a unique fixed size sequence, such as the widely used MD5 and SHA hashing algorithms. The problem with hashes is that a slight change in input can cause an avalanche effect and drastically changes the output. This will result in a unique hash for a slightly different payload. This result is unacceptable for a system that requires the estimation of the similarity between network streams/flows. Thus, similar flows will be marked by the system as entirely separate. Fuzzy hashing or context-triggered piece-wise hashing (CTPH) solves this problem. Fuzzy hashing applies traditional hashing to file segments. These segments are identified by a rolling hash algorithm. This algorithm produces a series of

	MD5	CTPH
Before	1d238b19da513ce35e129e7dc0169ad	3:Pg/vmNKzug:Y/vmNKzug
After	fe1b01ed311cd84e49a6b397d0e3e19	3:Pg/vmNKzul6A4jFS:Y/vmNKzulrr

Table 4.1: Comparison of MD5 with CTPH

'reset points' depending on the context of the plain text. The segments created before and after these reset points are then processed to generate hashes. The final hash is represented as a colon separated value. Thus CTPH makes use of both traditional and rolling hashes to create a segmented hash. It involves the ability to compare two distinctly different items and determine a fundamental level of similarity (expressed as a percentage) between the two (Ninja, 2007). This technique was used to develop the "Spamsum" program and originated as an effort by Dr. Andrew Tridgell (Tridgell, 2002) to find commonality between spam email messages. The payload hashed with this technique is added to the profile.
Example:
Contents of file alphabet.txt: "ABCDEFGHIJKLM"
After Modification: "ABCDEFGHIJKLM Edited by Fahim"

The difference in hash can be illustrated in Table 4.1

Discussion and limitation For this approach, the network profiles would comprise of the Spamsum hash of the original stream. This will reduce the network profile size considerably, as it will now be represented by a one line hash or fingerprint. String metric algorithms would then be applied on these profiles to determine the similarity between them. Another novel add-on to this method is the addition of the native string edit distance algorithm used by Spamsum for similarity measurement. Spamsum utilizes an edit distance algorithm to determine the similarity between any two given hashes.

Problems for calculating the Spamsum edit distances may arise if any two signatures use different 'reset frequency' or 'block sizes'. This block size mismatch may occur due to varying file context and size. This can result in a worst case scenario of the two files not being able to be compared, resulting in maximum dissimilarity. A contrived example of this can be a case of reverse input strings being undetectable by the algorithm.

Test 2: Spamsum with Reverse string Spamsum is used to scale down the variable length packet/stream payloads being tested. A detailed account of this was discussed in the previous section 4.3.2.2. From the literature, it was revealed that Spamsum is also equipped with an approximated string edit distance algorithm, which is used to determine similarity between any two Spamsum generated hashes. This algorithm uses dynamic programming and a customized cost/weight based edit distance algorithm to calculate the weight of each string operation. In Spamsum, the edit operations are weighted as: 'insert' and 'delete' edit operations are assigned a weight of 1, while each substitution is given a weight of 3 and each transposition a weight of 5. Spamsum edit distance score is scaled to produce a range between 0-100, where 0 means mismatch and 100 means 100% match or perfect match. This algorithm is incorporated into our methodology as it is customized to work well with Spamsum generated hashes Kornblum (2006).

A possible limitation, which can cause Spamsum to fail could arise in situations where the two hashes or signatures use different block sizes or 'reset frequency', or whenever the two strings are very different. In such situations, the score is automatically set to 0 or no similarity. For our domain, one such situation could arise when the payload is scrambled to an extent that it is reversed. This can be a strategy used by attackers to evade detection. A similar example or simulation is discussed in the NCD section 4.3.2.3, where we have two strings 'strA' and 'strB':

strA = "aaabbbcccdddeeefffggghhhiiijjjkkklll mmmnnnooopppqqqrrrssstttuuuvvvwwwxxxyyyzzz"

strB = "zzzyyyxxxwwwvvvuuutttsssrrrqqqpppooonnnmmm lllkkkjjjiiihhhgggfffeeedddcccbbbaaa"

Here strB is the reverse of strA. NCD (discussed in Section 4.3.2.3), compared to other string metrics, is able to detect a high similarity between the two strings, as discussed in 4.3.2.3. Using this same contrived example for Spamsum we find that Spamsum fails to detect any similarity between the two strings.:

Example

```
## import Spamsum module
>>> import Spamsum

## string variable strA and strB carrying the strings
>>> strA = "aaabbbcccdddeeefffggghhhiiijjjkkklll
mmmnnnooopppqqqrrrssstttuuuvvvwwwxxxyyyzzz"
>>> strB = "zzzyyyxxxwwwvvvuuutttsssrrrqqqpppooonnnmmm
```

```
lllkkkjjjiiihhhgggfffeeedddcccbbbaaa"

## Spamsum of strA and strB
>>> suma = Spamsum.Spamsum(strA)
>>> sumb = Spamsum.Spamsum(strB)

## print the two Spamsum variables to see Spamsum generated hashes
>>> print suma
3:te9DBClNO+fJ4I2FtXRWhFG6LdAv:2dQnSBA7m
>>> print sumb
3:vcv6SU/tU/Vs/8lO3dA3Bgnkt:EhKUmWMIWW

## Spamsum edit distance on hashes
>>> Spamsum.match(suma,sumb)
## No match here
Spamsum similarity score = 0
```

Here the similarity score of 0 indicates no match, whereas 100 means a perfect match.

Discussion The example above illustrates, that though Spamsum might be a good similarity metric, it has some limitations. The limitations can however, be removed using some special strategies on the input for this algorithm, thus pre-process the input to detect some variations, which were otherwise undetectable by Spamsum. One such method was discussed in the previous section 4.3.2.2, however, the results are not as satisfactory.

It is assumed that that there exists a means to determine similarity between reverse strings with more confidence, but that will involve having a reference string containing segments from both the original string (strA) and its reverse (strB). This will increase the chances/probability of having common segments or sub-strings present in scrambled strings or variants of the same or original string. This idea is illustrated with the example below:

Example

```
## Continued from above workspace

## String variable strC containing original and reverse string
```

```
>>> strC = strA + rev(strA)
>>> strC = "aaabbbcccdddeeefffggghhhiiijjjkkklllmmmnnnooopppqqq
rrrssstttuuuvvvwwwxxxyyyzzzzzzyyyxxxwwwvvvuuutttsssrrrqqqpppooo
nnnmmmlllkkkjjjiiihhhgggfffeeedddcccbbbaaa"

## Spamsum hash generated for this string
>>> sumc = Spamsum.Spamsum(strc)
>>> sumc
'3:te9DBClNO+fJgZtXRWhFG6LdAXud6SU/tU/Vs/8DuOU3dA3Bgnkt:2dQnYBA7zKUmoyIWW'

## Comparisions made with strA and strB
>>> Spamsum.match(suma,sumc)
53  ## or 53% similarity
>>> Spamsum.match(sumb,sumc)
49  ## or 49% similairty
```

Here we take a variable string C or 'strC' which contains the normal string 'strA' and appends its reverse variant 'strB' as a suffix. Next we construct the Spamsum hash for this new string 'strC' and finally compare it with both 'strA' and 'strB'. Interestingly, Spamsum with reverse string or $Spamsum_{rev}$ was able to identify matching sub-strings and is confidently able to identify approximately above 50% similarity. This approach is better than the one mentioned earlier in 4.3.2.2, and comparable to that used by NCD that is discussed in the next section.

4.3.2.3 Similarity with Normalized Compression Distance (NCD)

Normalized Compression Distance or (NCD) is an information theoretic measure that provides an approximation of the relative information distance between any two objects or strings that contain information in them. NCD is discussed later in section 4.3.2.3. Before understanding the internal working of NCD, it is imperative to understand its relationship with information theory and particularly Kolmogorov Complexity (KC). Additionally, the rationale for using compression as a means for measuring distance is also discussed here.

Exploiting redundancy with Compression Compression is the encoding of information using fewer bits. A compressor or source coder is used to compress information, which reduces the size of the original message to yield a compressed message. This message is recoverable using a decompresser or decoder. There are many types of compressors that use various algorithms for compression. Categorizing these algorithms based on encoding schemes, we have lossless and lossy compression algorithms. *Lossless compression* allows the flawless reproduction of the input data from the compressed stream, while *Lossy compression* discards some information, and thus minimizes the amount of data used in compression and decompression. Lossless compression is required for most computer text or data files. However, for images, video and sound, where it is difficult for humans to distinguish the change in quality, due to the perception caused by limitations in their sight and sound stimuli, lossy compression is used.

Text or lossless compression algorithms can be classified into two main categories:
1. Static, such as Huffman codes and
2. Adaptive, like Lempel-Ziv based compression.

The quality and efficiency of a compressor depends on the percentage by which it can reduce the message (in bits) in linear time. Hence, compressors are evaluated by their execution speed and compression ratios. Repeated occurrences of the same symbol or string would compress better, by similar amounts, as they have similar information in them. Hence, compressibility can be used to measure similarity between strings.

NCD uses compression for approximating the Kolmogorov's Complexity to determine the relative information distances between network streams. Our implementation of the compressor is based on zlib (Gailly and Adler, 1995), which uses an LZ77-type (Winters et al., 1996) algorithm for fast and efficient compression. Choosing the right compressor for the job is always tricky. It has been observed that an efficient compressor like bzip, might not be suitable for compressing small text files, as it adds more fixed size overhead to the file and hence increases its size. Since we are involved with compressing hashes of payloads, which preserve similarities, zlib which works well with small files was found to be the best tool for the job.

Kolmogorov complexity and Entropy In simple terms, Kolmogorov Complexity K states that the complexity of a string can be given by its shortest representation. Hence it is a measure of the descriptive complexity within an object. It refers to the minimum length of a program that will compute a string x and terminate (Li and Vitanyi, 2008). Compared to Entropy, the Kolmogorov complexity is a measure of the amount of complexity of a string, whereas entropy is the measure of uncertainty. Kolmogorov Complexity determines how "random" a string or message is, whereas entropy determines how random an entire distribution of messages are. Kolmogorov

Complexity can be described as follows:

$$K_{\varphi}(x) = \min_{\varphi(p)=x} l\,(p) \tag{4.5}$$

where p represents a program, φ represents a universal computer, and x represents a string. $\varphi\,(p)$ is the output of a program p by φ and $l()$ is the function to calculate the length.

Why NCD? Since Kolmogorov complexity is non computable in the Turing sense, an approximation using a compressor C is given in the form of the Normalized Compression Distance (NCD) Cilibrasi and Vitányi (2005). NCD is given by the formula:

$$NCD_{(x,y)} = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}} \tag{4.6}$$

Where x and y can be arbitrary text files.

Based on Kolmogorov complexity, the NCD provides a measure of similarity between any two given strings or files based on their analogous information densities. The result is a value between 0 and 1. '0' indicates that the items being compared are very similar, whereas a result of '1' indicates being highly dissimilar.

Test 1: For scrambled information, the compressor in NCD exploits redundancies of symbols or information between any two given strings or files. This is how it outperforms some other string metrics in situations that rely on accounting for the number of substitutions, insertions and deletions required to transform one string into the other. Applying this concept to network security, we can say that packets may have mutations in the form of the same exploit, but different shellcode, or variants of the same exploit. Similarity scores based on compressibility versus substitutions, insertions, deletions (done by string metrics) are more deterministic for this domain. For example, consider two strings 'strA' and its inverse 'strB':

strA = "aaabbbcccdddeeefffggghhhiiijjjkkklll mmmnnnooopppqqqrrrssstttuuuvvvwwwxxxyyyzzz"

strB = "zzzyyyxxxwwwvvvuuutttsssrrrqqqpppooonnnmmm lllkkkjjjiiihhhgggfffeeedddcccbbbaaa"

Calculating the Levenshtein distance between these two strings, we get:

```
>>> levenshtein_distance(strA,strB)
>>> 78 (or 0.781 on a scale of 0-1 )
```

The value 78 on a scale of 0-100 roughly translates to 78% dissimilarity or 19% similarity, whereas the NCD result is:

```
>>> ncd(strA , strB)
>>> 0.5375 (since NCD uses the scale 0-1}\
```

Here 0.53 means approx 50% similarity. As the second string '*strB*' is simply the first string '*strA*' reversed, and there are multiple common repeated elements, the 50% similarity returned by the NCD evaluation seems more intuitively appropriate than the very low 19% similarity given by the Levenshtein evaluation. Therefore the NCD comparison would theoretically seem to be a better method for quantifying the similarities between two strings, or, in our case, malicious streams.

Test 2: NCD with Spamsum hashes. In the previous section 4.3.2.2, it was discussed how Spamsum was used to create hashes of the payload and similarity metrics were applied on those hashes. As an additional example, NCD is tested as a similarity measures on the Spamsum generated hash for the payload:

```
>>> strA = "aaabbbcccdddeeefffggghhhiiijjjkkklllmmmnnnoooppp
qqqrrrssstttuuuvvvwwwxxxyyyzzz"
>>> strB = "zzzyyyxxxwwwvvvuuutttsssrrrqqqpppooonnnmmmlllkkk
jjjiiihhhgggfffeeedddcccbbbaaa"
>>>
>>> hasha = Spamsum.Spamsum(strA)
>>> hashb = Spamsum.Spamsum(strB)
>>>
>>> ncd(hasha,hashb)
0.8333333333333334 or approx. 17% similarity
```

Conclusion: This shows that applying NCD on the Spamsum hash of packet payloads does yield some similarity, but compared to NCD on the original stream (plain text or raw payload) it yields a worse score and has no advantages. It is concluded that NCD on raw payload performs similarity measurement with more confidence and hence is a better measure.

Conclusion and limitation

These information theoretic metrics provide a unique set of similarity measurement mechanisms. From a theoretical point of view, NCD has the capability of exploiting redundancies that exist in information contained inside packets, which gives it an advantage over other edit distance type string metrics. The edit distance algorithms provide an account of the exact measure of similarity expressed as a numerical value, which may be more accurate than the approximations provided by NCD, but is perhaps difficult to scale as the string size increases. Therefore, NCD is better than edit distance based string metrics.

The bursty nature of network traffic results in variable length packet/stream sizes. Network profiles extracted from such streams can pose a challenge for the string metric algorithms, as in many cases the number of insertions or deletions may increase exponentially. This will significantly effect the overall measurements. There is a need to scale down or normalize the network profiles before applying similarity measurements to them. Spamsum would perform well with its context triggered piecewise hash algorithm. The packet profile would be broken into smaller segments and each segment would be hashed and thus preserve the information in it. Spamsum, like NCD, has advantages over plain edit distance based measures, but would fail to provide an approximation in situations where the block sizes differ significantly, as disscussed in the reverse string test. Special pre-processing strategies are required to cope with this problem to make Spamsum measurements reliable.

At this stage it is difficult to suggest which information theoretic metric has the advantage. Both NCD and Spamsum use different strategies to calculate distance scores and both rely on computationally intensive operations like compression and hashing to meet their goals. Further testing is required to conclude clearly as to which metric has advantage over the other.

4.3.3 Hybrid approach

In the previous section 4.3.2.2 it was concluded that using Spamsum with the appended reverse string can be used to determine the similarity even for reverse strings variants with confidence. This means that in order to compare any two given strings ‘strA’ and ‘strB’, we first need to append their reverse to them, before sending them as input to the Spamsum sub-system for similarity measurements. Another interesting thing learned from this experience is that the resulting similarity score output is very close, or approximately equal to that of NCD. A question that arises here is: Are Spamsum and NCD closely related metrics for similarity measurement?

Hypothesis

In order to answer this question, the problem was studied both theoretically and empirically. In theory, both NCD and Spamsum use different means to determine similarity. NCD is based on compression and compressibility at its core, while Spamsum uses the context-triggered piece-wise hashing algorithm, along with a string edit distance algorithm. From the initial experiment, the two techniques did show similar results, but we need to test this on a slightly larger dataset to determine any correlation between them.

Methodology and Results

The hypothesis was tested on a small dataset of 500 samples of labelled malicious and benign streams. It was observed that Spamsum with reverse string appending ($Spamsum_{rev}$) and NCD are highly correlated, with a Pearson product-moment correlation coefficient of $r = 0.96$ and coefficient of determination $R^2 = 0.93$. However, these scores do not tell the whole story, and it was observed that in some circumstances only one of the two metrics identified similarities better over the other (see section 4.5. Based on these observations, it was decided to use them both by taking the minimum score of the two as the criteria to determine similarity. Hence similarity is now calculated as:

$$\text{Similarity Score} = \min(NCD(x, y), Spamsum_{rev}(x, y))$$

4.4 Experiments

In this section, experiments are conducted using similarity metrics to detect the similarity between variants of malicious network streams.

Distance Metric	D(strA,strB)
Levenshtein	0.78
Damerau Levenshtein	0.77
Hamming	0.78
Jaro Distance	0.51
Jaro-Winkler	0.5
NCD	0.53
NCD with Spamsum	0.83
Spamsum	1
Spamsum Reverse	0.53

Table 4.2: Comparison of Similarity metrics: Levenshtein, Damerau Levenshtein, Hamming, Jaro Distance, Jaro-Winkler, NCD, NCD with Spamsum, Spamsum, Spamsum Reverse using Strings: strA and strB. Scaled such that 0 means perfect match and 1 means complete miss match

4.4.1 Experimental Objectives

1. To simulate variation, by adding random fuzz to reference packet data to synthesize variants. (Discussed in: 4.4.2)
2. To measure the variation in the synthesized variants with respect to reference or original data. (Discussed in: 4.4.2)
3. To test the hypothesis of measuring network packet variants on real packet data. (Discussed in: 4.4.3)

4.4.2 Experiment 1: Simulation

A fuzzer script was created using Python[1] and the Scapy[2] module. This script takes as input any worm packet instance, creates N variant instances of the same packet, with K bytes of random fuzz replacing the original packet payload content per instance, resulting in a total of $K \times N$ replacements. Here fuzz refers to randomly chosen alphanumeric ASCII or hex characters. The script was tested with an original instance of a slammer worm packet. Three datasets of 50, 100 and 200 packets respectively were synthesized in PCAP format. Here each instance added 10 bytes of fuzz at random locations in the payload to its predecessor. All these packets are created as sequential increments of their predecessor. Next, we tested each dataset for detection using Snort (Roesch et al., 2006) and then with our proposed system (Using methodology discussed earlier, see 4.3.2.2). Results are summarized in Table 4.3. For Slammer worm detection, Snort was configured with the following signature:

[1]http://www.python.org/

[2]http://www.secdev.org/projects/scapy/

```
alert udp EXTERNAL_NET any ->  HOME_NET 1434
(msg:"SQL version overflow attempt" ; flow:to-server; size:> 100;
content: "|04|"; depth:1; reference:bugtraq,5310; reference:cve,
2002-0649; reference: url, www.microsoft.com/technet/security/
bulletin/MS02-039.mspx;classtype:attempted-admin; sid:2050; rev:15;)
```

A distance threshold is setup as a cut-off value for similarity. For the threshold we require a moderate value, that will not incorporate too many false positives and yet should be tight enough as not to miss any worms. Based on prior experimentation and work, we chose the NCD threshold value to be 0.57. This means that any packet which shows a similarity score of less than 0.58 with the original worm packet will be considered malicious.

As a comparison, this same experiment was then repeated using various similarity metrics discussed earlier. (see section 4.3).

4.4.3 Experiment 2: Mixed Internet and worm traffic

Another experiment was designed, in which the system is tested with another original PCAP dataset comprising of mixed Internet traffic. This was a two stage experiment:

In the first stage, a dataset was created from packets extracted from various malicious and benign pcaps collected from wireshark.com and pcapr.net. A total of 66 packets were extracted from these pcaps comprising of 30 worm packets, 10 Netbios Packets, 10 POP3 packets, 6 SMTP packets and 10 HTTP packets.

For the second stage we collected and labelled 500 malicious and benign streams comprising of categories including: TFTP download, Buffer Overflow (LSASS), Shellcode, TFTP download , Oracle DB Connection Attempt, Remote Framebuffer Protocol (VNC) Version String 005, Big Yellow (Symantec) Buffer Overflow Exploit, TrendMicro ServerProtect Buffer Overflow Exploit, FTP download (via cmd.exe), Webmin Exploit GET /unauthenticated/..%01/..,RPC/DCOM Buffer Overflow Exploit with simple polymorphic random data, ASN1 Buffer Overflow Exploit (Shellcode 1), MySQL UDF Injection and Execution with FTP download, MySQL root access, SMB/CIFS Scan, HTTP GET Request (absolute http Path), HTTP GET Request /w00tw00t.at.ISC.SANS.DFind:), HTTP OPTIONS Request with User Agent Infos, POP3 Login Attempt, SIP scanner,ack,invite,cancel , MS-directory service (port445), NMAP SERVICE PROBE, MMS stream, EPMAPPER (port 135) and PROPFIND WEBDAV. The system is also tested on this dataset and results are discussed in 4.5.3.

In order to conduct the experiment, a reference class is selected from any of the classes

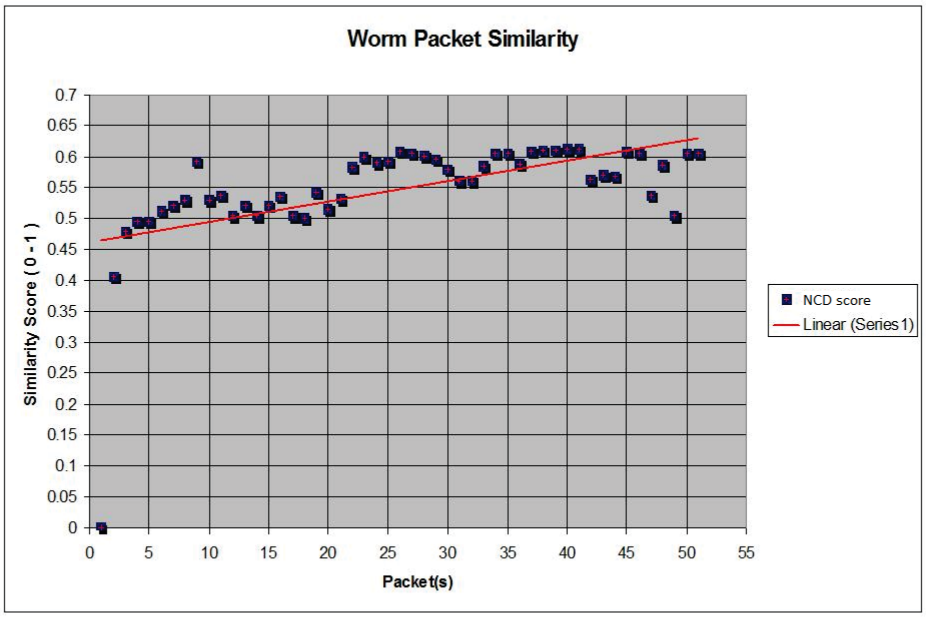


Figure 4.1: NCD results for 50 fuzzed packets. 50 packets were synthesized from an original slammer worm packet instance. Each synthesized packet contained 10 bytes of random fuzz added to its predecessor instance in a cumulative manner.

mentioned above. Next, similarity scores are calculated between this reference class and all the other classes and samples in the dataset. Results of each similarity metric used is written out to a file for analysis. An analysis is carried out to identify any false positives or negatives due to overlaps. Ideally, we want to find the metric which provides the best means to partition the reference class from other classes in that dataset.

4.5 Results

A detailed analysis of the results obtained from the experiments discussed earlier are provided in this section:

4.5.1 Result 1: Simulation Results

Graphical plots of the NCD results for experiment 1 are illustrated here in Figures 4.1, 4.2 and 4.3:

Figures 4.1, 4.2, 4.3 for experiment 4.4.2 suggests that as the number of fuzz packets increases, the Snort detection rate goes down. For our proposed IDS system the general trend is similar, but with a few exceptions, wherever the fuzzer added such symbols to the payload, which were similar to the replacements, it increased the overall similarity

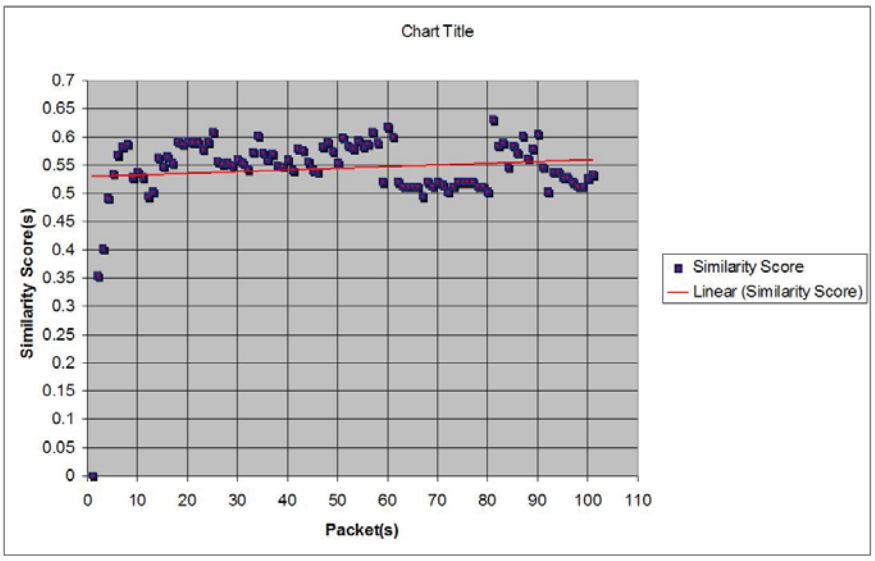


Figure 4.2: NCD results for 100 fuzzed packets. 100 packets were synthesized from an original slammer worm packet instance. Each synthesized packet contained 10 bytes of random fuzz added to its predecessor instance.

N (instances)	K bytes (fuzz)/pkt	KxN (replacements)	Snort Detection %	Detection by our Sys. %
50	10	500	45	47
100	10	1000	19	52
200	10	2000	10	40

Table 4.3: Comparison of IDS: Snort vs Similarity metric

score, thus making all such packets more detectable by our system. This may lead to statistical errors and further experimentation is required. This, to some extent, simulates the masking of information in packets by the attacker in the form of encryption by substitution. The NCD threshold value serves as a key for our system. The tighter or more closer to 0 the value is, the lower the detection and False positive (FP) rates. Further experimentation will help us to determine a suitable threshold value that will be more effective. For a test threshold value of 0.55 our system generated approx 60% detection rate for the 200 sample pcap, approx 52% detection rate for the 100 sample pcap and 47% for the 50 sample pcap, compared to 10%, 19% and 45% respectively for Snort, as shown in table 4.3.

Repeating this same experiment using different similarity metrics, we get some interesting results:

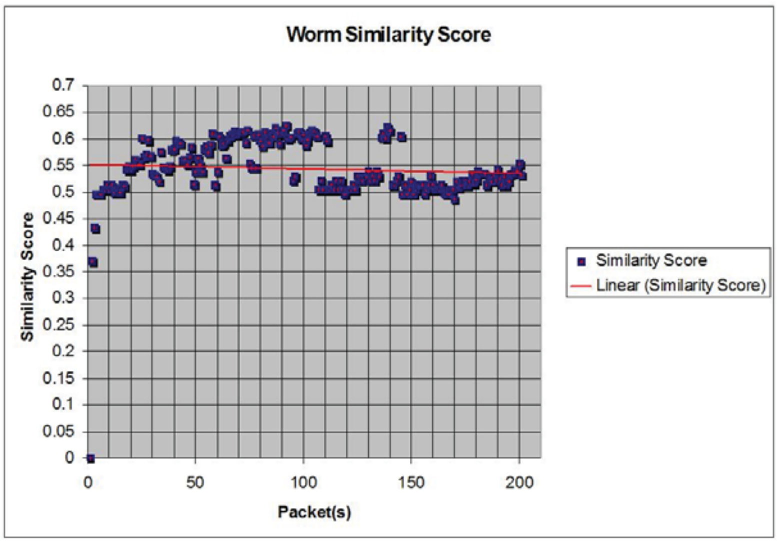


Figure 4.3: NCD results for 200 fuzzed packets. 200 packets were synthesized from an original slammer worm packet instance. Each synthesized packet contained 10 bytes of random fuzz added to its predecessor instance.

Comparative results

Comparative results for all similarity metrics are shown in figure 4.4 and condensed in Figure 4.10. Theoretically, there should be a gradual increase in distance, as the amount of fuzz in the packet variants increases. This should be enough to draw decisive boundaries and avoid overlaps.

Here it can be seen that as the amount of fuzz in packets increases, their similarity starts to decrease almost linearly. For edit distance based string metrics: Levenshtein, Damarau and Hamming distances exhibit this trend in the form a steep curve, surpassing the scale of 0-1. Next the Jaro and Winkler distances show a slight increase in this trend, but the line is quite flat indicating that the effect of fuzz is not affecting the similarity too much. This to some extent causes ambiguities as it will become difficult to draw decision boundaries between what is or is not similar.

NCD seems to conform to the norm by exhibiting an expected behaviour in terms of a gradual increase limiting by distance scale 0-1. For NCD, a similarity threshold of 0.8 can be used as a decision boundary in this case. Applying NCD on Spamsum hashes for this experiment reveals some interesting findings. The similarity scores suggest a sharp increase in similarity, followed by a slight dip and then a flat line. This again is ambiguous behaviour in terms of similarity measurements. Once again it becomes difficult to draw a decision boundary here, especially for example, for a threshold of 0.8, all packets suggested by the flat line will be considered similar by the system. This

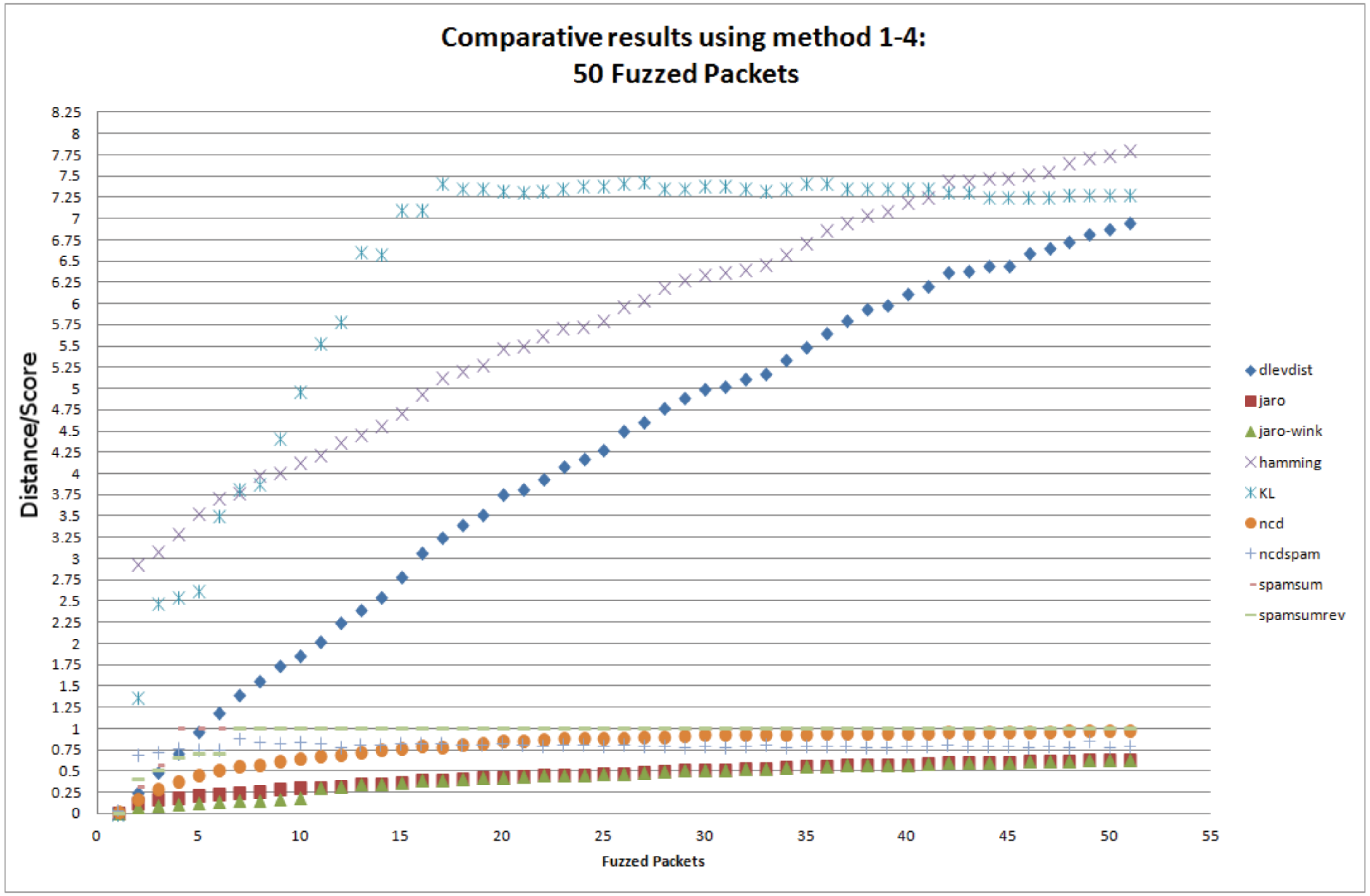


Figure 4.4: Comparative results using various similarity metrics (both edit distance and information theoretic) on the 50 fuzz packet dataset. The y-axis shows distance or similarity scores without normalization. The x-axis represents each of the 50 fuzzed packet.

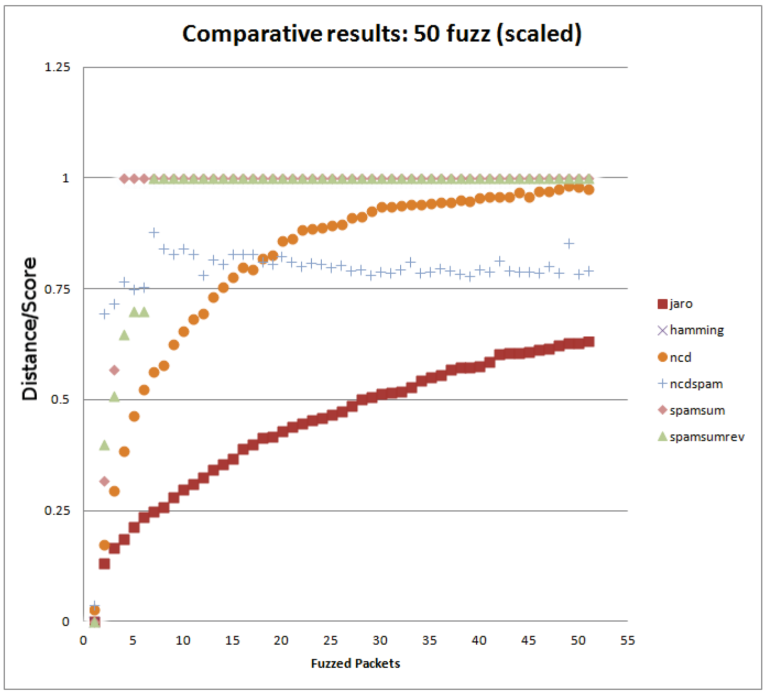


Figure 4.5: Comparative results on the 50 fuzz packet dataset (similarity scores are normalized and scaled between 0-1). Here 0 = identical and 1 = dissimilar

is against the expected ground truth. Hence using this method a 100% fuzzed packet will still be considered similar.

Spamsum with reverse hash is able to detect more similar packets correctly compared to Spamsum on the original sample, but fails to detect the trend over the subsequent packet as done by NCD. This further strengthens the hypothesis laid down in the hybrid similarity metric discussion, see 4.3.3, where the need for a hybrid approach is advocated. From this experiment we can now conclude that Spamsum with reverse hash (4.3.2.2) and NCD on the original sample (4.3.2.3), which are combined to create a hybrid technique (discussed in 4.3.3), outweighs other similarity metrics, as comparatively, this methodology is able to detect more similar packets correctly.

A similar trend was observed with 100 and 200 fuzzed packets.

Sensitivity test:

In order to avoid any statistical bias that may have been introduced due to incremental iterations of fuzz resulting in packets with cumulative fuzz as stated above, another experiment was designed. Nine original packets of the Slammer worm instance were selected. All these instances were tested and verified by Snort detection. Next for each

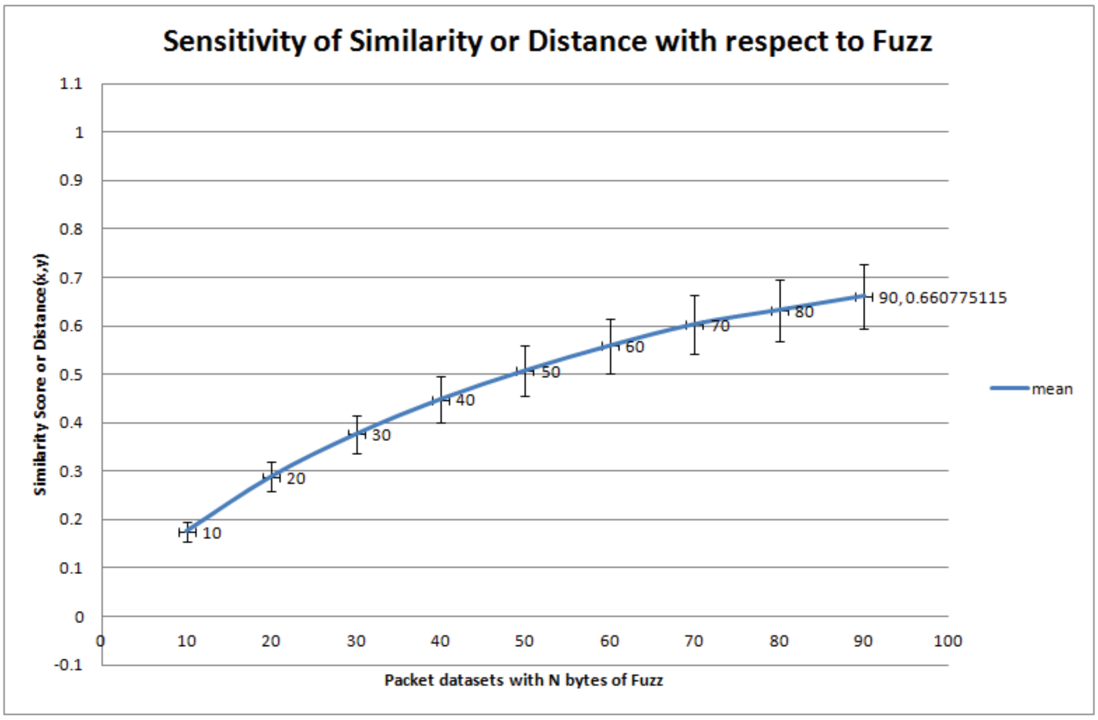


Figure 4.6: Sensitivity test of distance with incremental fuzz

original packet instance, 100 variation were created/synthesized from the reference instance with varying amounts of fuzz. This resulted in groups of packets based on incremental fuzz mutations. Here, for example, the first group contained 100 packets synthesized from the original reference, created using 10 bytes of fuzz mutations per packet, the second group was created using 20 bytes of fuzz and so on, with the ninth group having 90 bytes of fuzz in each instance.

For each group a distance score was calculated between the original reference and the tainted or fuzzed or mutated packet. Next the Mean and Standard deviation across all these scores was calculated for each group and the results were plotted on a graph shown in Figure 4.6. Here a linear positive slope is observed, which conforms to the theory that as the amount of fuzz or mutations increase in packets, their distance from the reference increases.

Conclusion: It can be concluded that this methodology can be used to reliably identify variants of packets, worms or malicious streams.

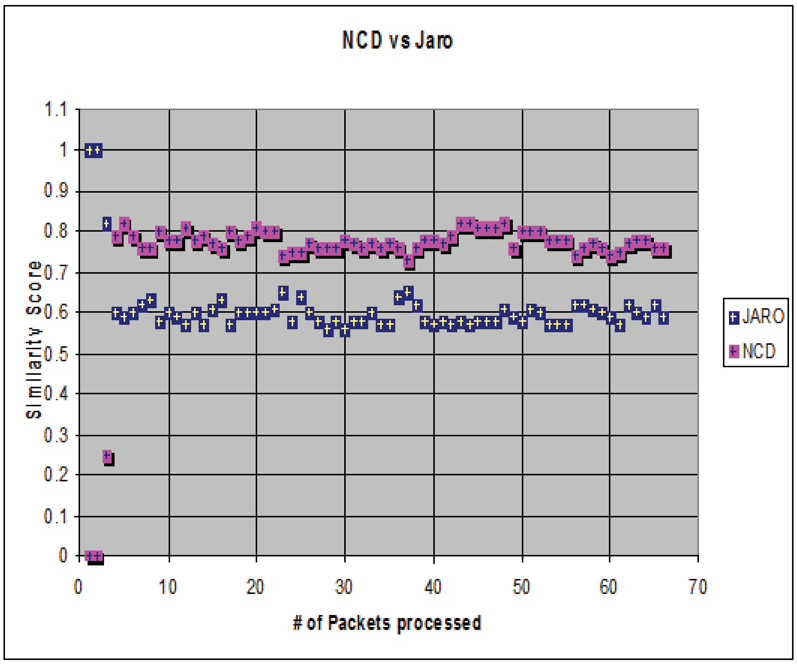


Figure 4.7: NCD and Jaro results for fuzzed Slammer worm packets

4.5.2 Validation of Simulation

In this section we validate the rationale for using similarity metrics to detect the similarity between variants of malicious network streams.

In order to validate the results obtained by various string metrics and semi-metrics, edit distances like Jaro-Winkler Distance, Hamming Distance, Levenshtein distance, and information theoretic measures like Information gain or Kullback-Leibler divergence and Entropy have been used on the 50 packet dataset as shown in Figure 4.10. A general relationship has been identified between all string metrics using entropy. For this dataset it has been observed that packets with higher entropies than the reference packet yield a higher or worse similarity score. We believe this is due to the increase in the complexity of the string. The added randomness decreases the similarity of the string and hence that much more energy is required to transform it into the reference.

4.5.3 Result 2: Mixed Internet Traffic

Comparative results of one instance of the Slammer worm with the dataset discussed in 4.4.3 are shown in Figures 4.10 and 4.7. There were only 3 packets of the Slammer worm instance in this dataset (Packets 1 - 3), which show a very close similarity score, while other packets are distinctly dissimilar.

The results from the second part of this experiment are shown in Figure 4.8. This experiment was carried out on a larger dataset of 500 samples. We observed that NCD outperformed other metrics, by providing a clear class boundary between samples of the reference labelled class and other labelled classes in the experiment. For NCD a threshold of 0.75 can correctly identify similar items from that class, without any false positives as shown in Figure 4.9. Jaro seems to be the second best, but still has ambiguities that will lead to false positives. For Jaro a threshold of 0.45 will cause false positives due to overlaps and anything below this threshold will cause false negatives. Spamsum with reverse hash failed to identify any similarity. This further endorses the need for having a hybrid approach (discussed in section 4.3.3)

4.6 Limitation

The method detailed in this chapter will have some limitations. An attacker could intentionally alter the entropy or complexity of the attack traffic or exploit to such an extent that the classifier fails to identify it correctly. This could be achieved by adding interleaved random symbols in with the actual exploit data. However, this may well affect the exploit and make the overall attack considerably more difficult. This technique would not work with evasion techniques involving substitution/encryption and traffic containing polymorphic version of the same malicious stream. However, malware containing polymorphic binaries should be extracted from such streams and tested separately in a sandbox environment. A similar classification methodology could be applied but for in-memory code analysis on such samples.

4.7 Comparison with Inverse Compression Ratio

In the past, researchers have used information-theoretic approaches for intrusion detection. Approaches involving the use of entropy for the detection of anomalous flows suggested by Lakhina et al. (2005) and for detecting DDos attacks was suggested by Feinstein et al. (2003). Work done by Evans and Barnett (2002) used compression to compare the complexity of legal FTP traffic with illegal traffic, but at a header level only, whereas we use entire TCP sessions. Kulkarni and Bush (2006) also performed denial of service measures using compression ratios based on Kolmogorov complexity by estimating the entropy of 1's contained in the packet, which has some limitations. Eiland and Liebrock (2006) detected network scans from normal traffic using Kolmogorov complexity by estimating the inverse compression ratio of scan and normal traffic.

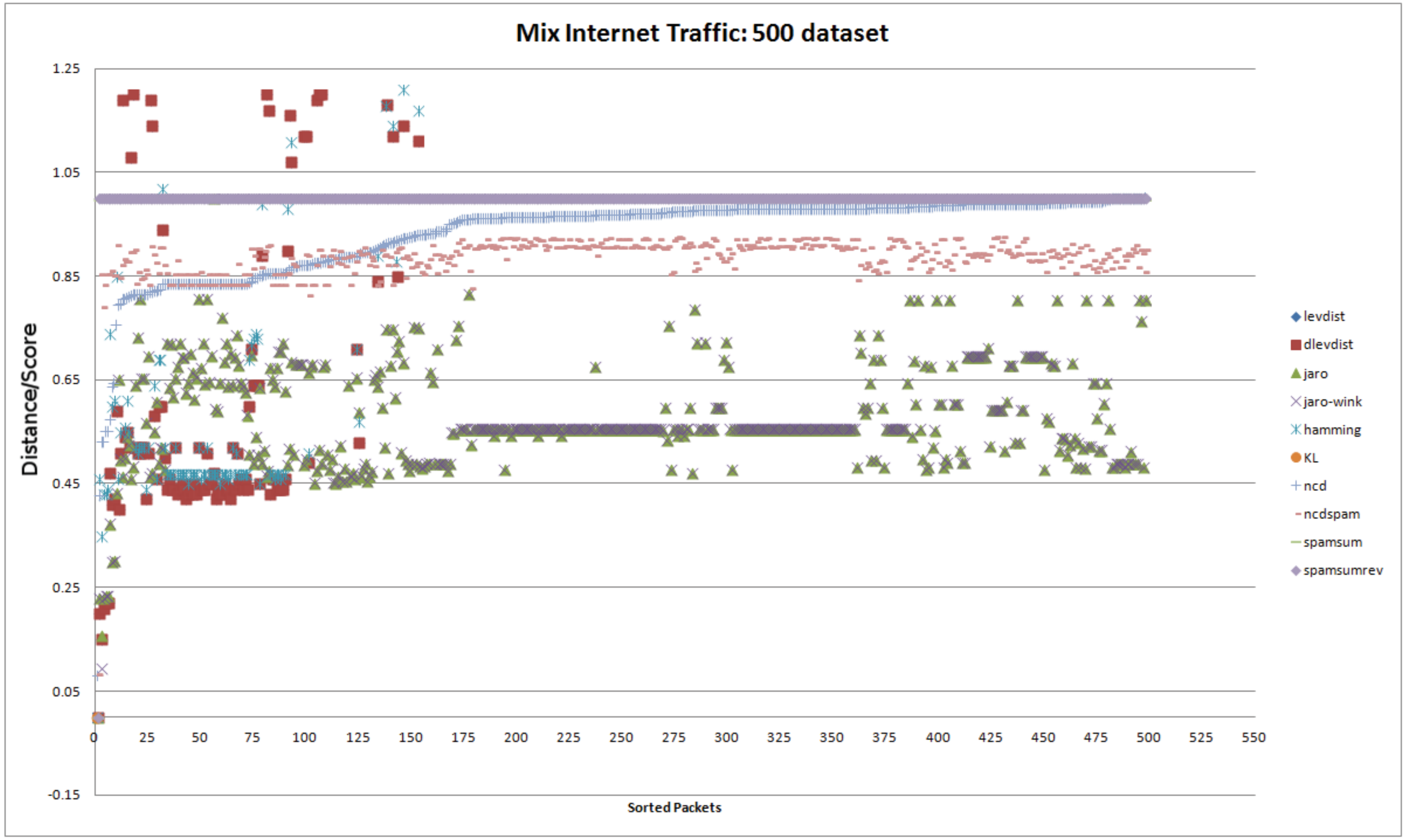


Figure 4.8: Mixed Internet traffic, containing malicious and benign samples from the 500 dataset compared with a reference Slammer worm exemplar/instance.

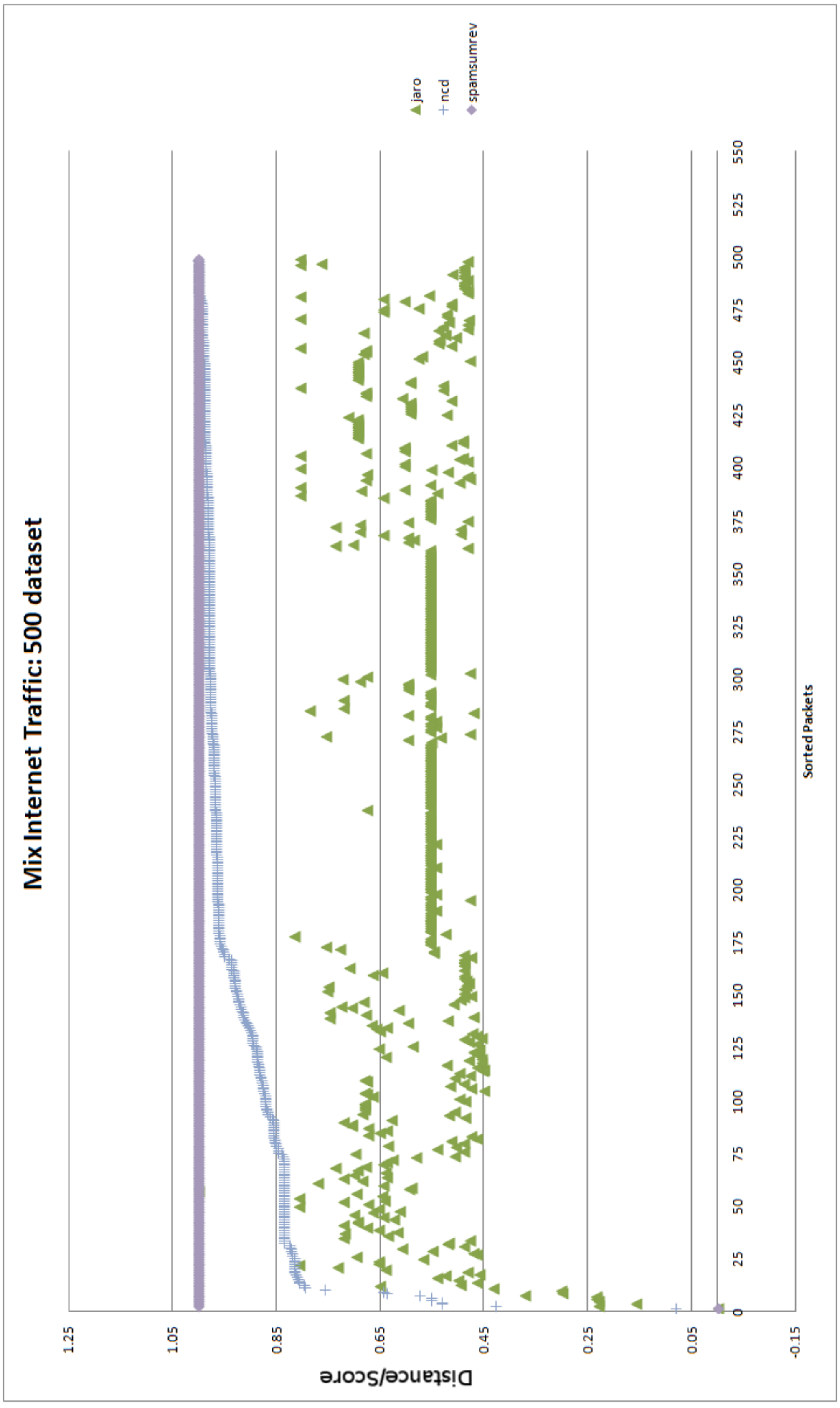


Figure 4.9: Condensed results: Mixed Internet traffic, containing malicious and benign samples from the 500 dataset

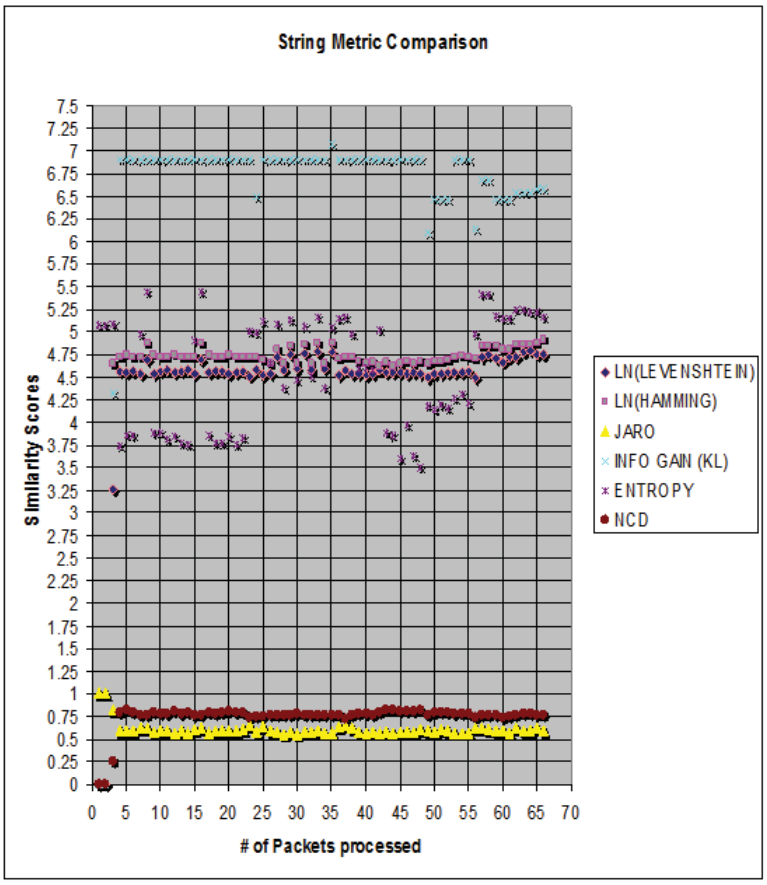


Figure 4.10: Comparison of similarity metrics on fuzzed worm packets, with reference Slammer worm instance

The technique proposed in this chapter for similarity measurement were compared with the aforementioned work including the work of Eiland and Liebrock (2006). They used inverse compression ratio to detect scan traffic. Inverse compression ratio (ICR) was used as it is bounded by zero and one and hence scales nicely, compared to compression ratio, which ranges from one to infinity. Since inverse compression ratio is checking the ratio of compressed file with its uncompressed version, any file having more variations will not compress well compared with the one having fewer variations. Applying this methodology to our domain, exploits with variations will not compress better and it will be difficult to classify them as variants. To empirically prove this, we performed an experiment involving a small dataset comprising of 500 labelled samples from various classes and applied inverse compression ratio to all these samples. The ratios were then used to see if we could classify the samples correctly. We found that this technique fails to partition or classify various types of attack traffic that we have in our dataset. Variations within the same class were causing misclassification due to varying results from the ratio.

4.8 Conclusion

It can be concluded that the use of packet payload information as signatures to classify malicious streams using network data does work. We concluded that we can perform similarity measurements on packet payload data with confidence. Experimentally we found the system to be robust and can allow for significant changes in the signatures without misclassifying the stream. We found that for this type of data, NCD and Spamsum metric were the most appropriate similarity metrics and concluded that since both are highly correlated and have some advantages over the other, we can use them in a hybrid approach for similarity measurements.

Chapter 5

Clustering and Classification for Intrusion Detection

"There are known knowns; there are things we know that we know. There are known unknowns; that is to say there are things that, we now know we don't know. But there are also unknown unknowns there are things we do not know we don't know"
–Donald Rumsfeld, US Secretary of Defence

The previous chapter demonstrated how the use of edit distance and information theoretic similarity metrics can help determine similarity between network profiles for intrusion detection. Several similarity metrics were reviewed and a few were chosen from the literature, and used to determine the similarity between various packet or stream profiles. Some preliminary tests have been conducted to identify the most suitable metrics for our domain. Different combinations for measurements were discussed and metrics of choice were proposed based on theoretical analysis and some initial experiments.

This chapter builds on this work and investigates the use of machine learning techniques for the classification of malicious network streams. This chapter discusses clustering and classification techniques using data obtained from similarity measurements of malicious network streams. Some machine learning approaches for intrusion detection are discussed. Experiments are designed to find the best clustering algorithm and cut-off threshold for our domain given the similarity metrics. The results are analysed with a discussion on ROC-AUC based measures and Accuracy. Results obtained by comparing all the clustering algorithms used for the evaluation will be discussed and the best ones highlighted.

5.1 Problem Description

In order to establish the best performing metric suitable for our domain, the metrics are compared and evaluated both theoretically and empirically, using statistical machine learning techniques. These techniques would help in correctly grouping the measurements obtained from similarity metrics into their respective groups for classification. To achieve this classification goal, it is necessary to find a combination of the best similarity metrics and clustering algorithms to use. The following questions are addressed in this chapter:

1. How to group similar items together?
2. How to determine the optimal threshold to partition data into groups?
3. How to find the best clustering algorithm(s) for our domain?
4. How to evaluate the classifier results for correctness?
5. How to identify the most suitable similarity metric for our domain using the above evaluations?

5.2 Background

5.2.1 Machine Learning

The discipline of machine learning provides a set of algorithms for solving problems that require learning, adapting or evolving a computer program based on empirical data. Examples or observations from the training data can be used to extract unique features or characteristics to create models. These models can be used to automatically learn or recognize complex patterns that emerge in the data or make intelligent decisions based on them. The performance of the learner is directly proportional to the coverage of features in the training data. Since not all features can be made readily available to the system during training, the learner should generalize known features in order to learn and identify new and novel features from the test set. In other words, the learner should be able to generalize from its experience.

5.2.2 Machine Learning for Intrusion Detection

Cyber criminals target individual and organizations' data for their gain. Security tools are deployed in the cyber infrastructures to defend against these cyber criminals and their ill intent. Information security tools like intrusion detection and prevention systems, anti-virus systems and firewalls produce large amounts of data in the form of logs. With all these security appliances generating large amounts of data, the industry is finding it hard to cope with and make appropriate intelligent decisions. Data mining, statistics and machine learning capabilities are required to address the challenges of cyber security i.e. to process vast amount of data. We intend to use machine learning techniques to extract knowledge from data for better intrusion detection and prevention. Machine learning techniques will give us the capability to extract strong patterns or derived rules from the data to group entities and to use that knowledge to predict new classes of data. It serves as an iterative process of extracting knowledge from data. Applying this concept to intrusion detection can help better detect intrusive behaviour in networks and systems, than done by researchers in the past.

Clustering and Classification

Machine learning algorithms are used for our domain, for the clustering and classification of malicious network streams. Instances of malicious and benign network traffic profiles are used as a template or an example to classify other similar instances, thus "learning from examples" (Aha et al., 1991), this is an example of supervised learning. An instance space consists of a set of stored instances. Given an instance from instance space, an instance-based learning algorithm is used to map instances to categories. Instance-based learning algorithms use similarity and classification functions for their operation. We hypothesize that instance-based machine learning algorithms with random instance selection, utilizing input from similarity metrics and clustering algorithms can provide good classification.

Classification is the process of identifying which set of categories an observation belongs to [1]. A similarity score is established between incoming packets or streams and the instances in the instance space. Classification is carried out using a model comprising of instances and their corresponding thresholds to categorize the input streams. A confusion matrix is generated as a result of this classification, populating table entries for the predicted class and the actual class, to evaluate the performance of the classifier. From this confusion matrix we calculate: Accuracy, Area Under the Curve (AUC) and finally generate a Receiver Operating Characteristics curve (ROC curve) to visualize the performance of the binary classifier.

[1]http://en.wikipedia.org/wiki/Classification_in_machine_learning

5.3 Evaluation

Evaluation measures play a crucial role in both assessing the classification performance and guiding the classifier modelling. Traditionally, Accuracy is the most commonly used measure for these purposes. Accuracy is defined as:

$$Accuracy = \frac{\#TP + \#TN}{P + N} \tag{5.1}$$

here:

$$P = TP + FN \tag{5.2}$$

$$N = FP + TN \tag{5.3}$$

However, for classification with the class imbalance problem, Accuracy is no longer the appropriate measure as the less represented classes can have very little impact on accuracy compared to the dominant class (Weiss, 2004). The area under the receiver operating characteristic curve or ROC-AUC based measures can be used to provide a better means to evaluate such imbalanced datasets.

The ROC curve provides a graphical plot between the true positive rate (TPR) or sensitivity (or recall) and the false positive rate (FPR) or 1-specificity (or true negative rate), here:

$$TPR = TP/P \tag{5.4}$$

$$FPR = FP/N \tag{5.5}$$

AUC[2][3] is defined as:

$$AUC = \int_0^1 ROC(t)dt \tag{5.6}$$

Here $t = (1 - specificity)$ and $ROC(t)$ is sensitivity.

Working with imbalanced classes, a variety of classification metrics could have been used

[2] `http://www.floppybunny.org/robin/web/virtualclassroom/dss/articles/sensitivity_specificity_accuracy_CI.pdf`

[3] $AUC = (1 + TPR - FPR)/2$

to evaluate our system. A few notable ones are: Receiver Operating Characteristics (ROC), cost curves, precision recall curves, precision, Q-score, F-score, recall and G-mean.

Precision measures the exactness of the positive labels among all positive labelled samples. Precision is sensitive to the distribution and does not indicate any FN. *Recall* measures the percentage of correctly labelled positive samples from all positive class samples. F-measure of F-score combines recall and precision and thus provides more details on the accuracy of the classifier. Several of these metrics are defined here:

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{5.7}$$

$$Recall = \frac{\#TP}{P} \tag{5.8}$$

$$F - measure = \frac{(1+\beta)^2 \cdot recall \cdot Precision}{\beta^2 \cdot Recall \cdot Precision} \tag{5.9}$$

Since compared to these metric ROC provides a means to compare different models and select the optimal one using cost versus benefit, we choose ROC as it gives an accurate idea of the classifiers performance.

5.3.1 ROC

The accuracy of a classifier can be evaluated using Receiver Operating Characteristics (ROC)(Fogarty et al., 2005) curve analysis as shown in Figure 5.1. ROC provides a good means to measure the trade off between true positive rate (TPR) and false positive rate (FPR) as seen by the classifier. The points on the curve represent the performance of the classifier. The ideal model is one that obtains a True Positive Rate of one and a zero False Positive Rate (i.e., TPrate = 1 and FPrate = 0, point A in Figure 5.1. A worst case scenario would be point B, coordinates (1,0), where TPR is zero and FPR is a maximum.

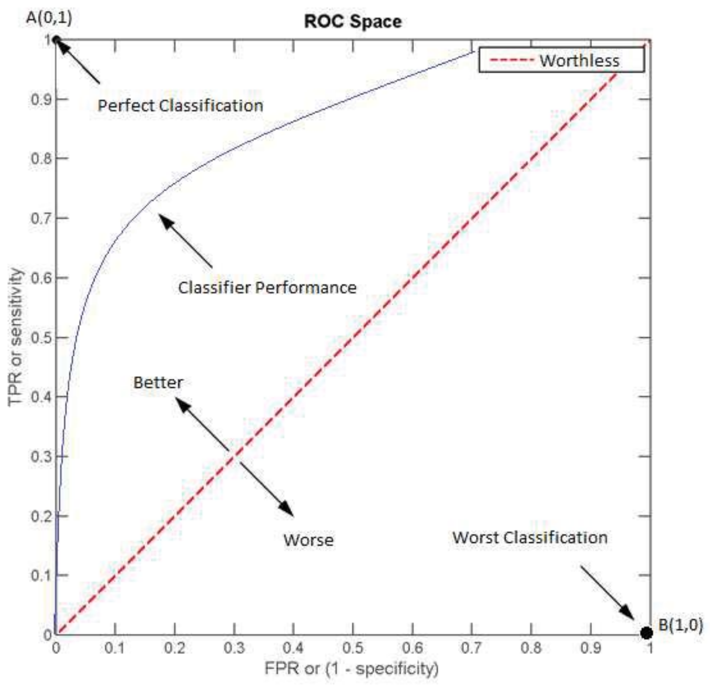


Figure 5.1: ROC Curve: the points on the curve represent the performance of the classifier. The ideal model is one that obtains a True Positive Rate of one and a zero False Positive Rate (i.e., $\text{TPrate} = 1$ and $\text{FPrate} = 0$, point A in this figure. A worst case scenario would be point B, coordinates (1,0), where TPR is zero and FPR is a maximum. A model that makes a random guess should reside along the line connecting the points (TPrate = 0, FPrate = 0), where every instance is predicted as a negative class, and (TPrate = 1, FPrate = 1), where every instance is predicted as a positive class.

5.4 Methodology and Experiments

5.4.1 Information visualization and analysis through Multi-dimensional scaling MDS

Before attempting any clustering or classification algorithms to identify clusters or groups in network data, the data is analysed carefully. From statistics we have multi-dimensional scaling or MDS, which is an information visualization technique for exploring similarities or dissimilarities in data. It provides an aerial view of the natural clusters that are forming in the data. This is done by reducing multi-dimensional data and visualizing it in 2 or 3 dimensions using the MDS algorithm. MDS provides a means to visualize how far away, or close together, the data points are in two dimensional space.

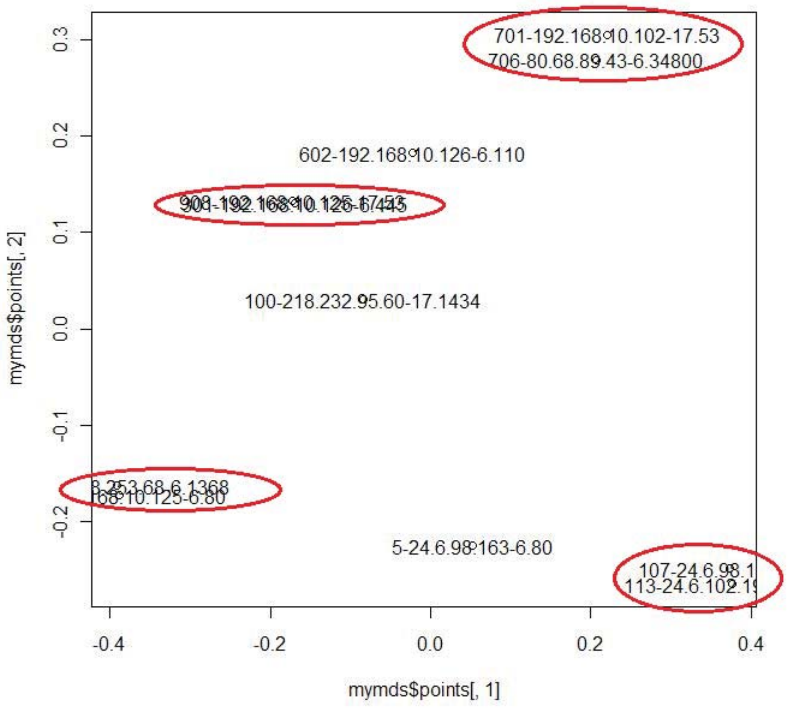


Figure 5.2: A 2D visualization of a small dataset of different types of packet data, using MDS. The natural clusters forming are highlighted and partitioning between different packets types is clearly visible.

Fifteen labelled packet samples are chosen from a small dataset comprising of 66 labelled packets of worm and benign packets (discussed in experiment 2 of the previous chapter). In order to determine how similar or dissimilar each packet is from the other, a pair-wise $n \times n$ similarity matrix of these 15 packets is created, using the minimum of NCD and Spamsum: $\min(NCD, Spamsum_{rev})$ as the similarity metric. This similarity matrix is a matrix of scores that represents how similar or far away in space each point in the dataset is to the other. Applying MDS on this small matrix and visualizing it, results in a graphic illustrated in Figure 5.2. Clusters formed between similar packets are highlighted in this figure. The empty space between the different types of packets expresses the partitioning of this type of data. This analysis serves as a preface which indicates that clustering techniques from machine learning can be applied on such data to exploit the inherent grouping that exists within it.

5.4.2 Experimental Objectives

We conducted a series of experiments to:

- Identify the best clustering algorithm. (Discussed in section 5.4.4)
- Identify the optimal cut-off or threshold value. (Discussed in section 5.5.4)
- Evaluate the correctness of the classifiers using confusion matrix and ROC curves (Discussed in section 5.5.5)
- Finally, based on the above results, identify the best similarity metric. (Discussed in section 5.5.5)

5.4.3 Preparing Test Datasets for experiments

For one of our experiments, ten small datasets extracted from a large dataset were created. Each dataset contained 500 malicious or benign streams randomly picked from the large dataset presented in the chapter 3. These samples comprise of the raw stream payloads, which are referred to as stream or network profiles. These labelled network profiles are in the form of a text file with the naming format of “c.a-the md5sum of the stream”, where ‘c’ is the class and ‘a’ is the subclass of the labelled stream. Five hundred such profiles were created using this methodology – one for each stream in the dataset.

In order to measure the relative similarity scores between these packet profiles, metrics such as: NCD, Levenshtein Distance, Hamming distance, Jaro distance, Spamsum and Hybrid were used. Edit distance metrics were used on raw profiles, as discussed in the previous chapter. Information theoretic measures like NCD and Spamsum with reverse string, were also discussed in the previous chapter. For each metric, a combinatorial similarity score was calculated across the dataset, resulting in 500×500 combinations per instance.

Both string metric calculations and Fuzzy hashing the payload are computationally expensive operations and they gravely affects the performance of the system, so pre-processing, automated-analysis and classification are performed offline. For this experiment, the system is designed to start its operation on receiving 500 streams in the buffer or wait for n seconds and process the buffer, whichever triggers first.

5.4.4 Experiment 1: Determination of the best Clustering and Classification algorithm

The similarity scores obtained via the similarity metrics are clustered, in order to group similar items together. Various clustering techniques are available in the literature. We choose the following based on their relevance with our domain. These clustering techniques are then tested with different similarity metrics and their performance is observed and compared. Some visualizations are presented in this section using a small dataset of 100 packets extracted from malicious classes (such as: MSSQL SLAMMER, CONFICKER, ZEUS, SADMIND, SMB ATTACK) and benign classes (such as:POP3, SMTP, HTTP, SMB), later visualizations are created using 500 packets.

Minimum Quartet Tree Cost (MQTC)

To construct a tree from a pair-wise distance matrix of similarity scores between objects, we utilize the tools provided by the freely available CompLearn toolkit (Cilibrasi, 2003). This tool makes use of a heuristic to implement the quartet method. The heuristic is called the standardized benefit score S(t). The quartet method proposed by this technique is the Minimum Quartet Tree Cost problem, which is an NP-hard graph optimization problem. A visualization produced using the Complearn toolkit on a dataset of 100 packets is shown in Figure 5.3.

Limitations: While effective, it was observed that the minimum quartet tree cost (MQTC)-based clustering(Cilibrasi and Vitányi, 2005) becomes highly inefficient (very slow to process a result) as the dataset size increases (Abbasi and Harris, 2010). For a small test set of 200 samples, the MQTC algorithm took 26 hours to process the data and produce a result. In our analysis other clustering methods were investigated, and compared with MQTC.

Unweighted-Pair Group Method with Arithmetic Mean or (UPGMA)

UPGMA is an agglomerative or hierarchical clustering method used heavily in bio-informatics and very well known for the creation of phenetic or phylogenetic trees. The motivation behind using UPGMA was to test its ability to create phenetic trees of labelled variations of malicious streams. Based on past observations (Abbasi and Harris, 2010), we set the threshold value to 0.55 and ran the system to cluster a relatively small dataset of 100 packets. The results were quite encouraging as can be seen in Figure 5.4. The rectangular boxes show the clusters, the dark oval nodes represent similarity and dissimilarity percentages, while the light coloured flat oval nodes represent the packets. In a very short time, UPGMA was able to correctly cluster the dataset according to its labels.

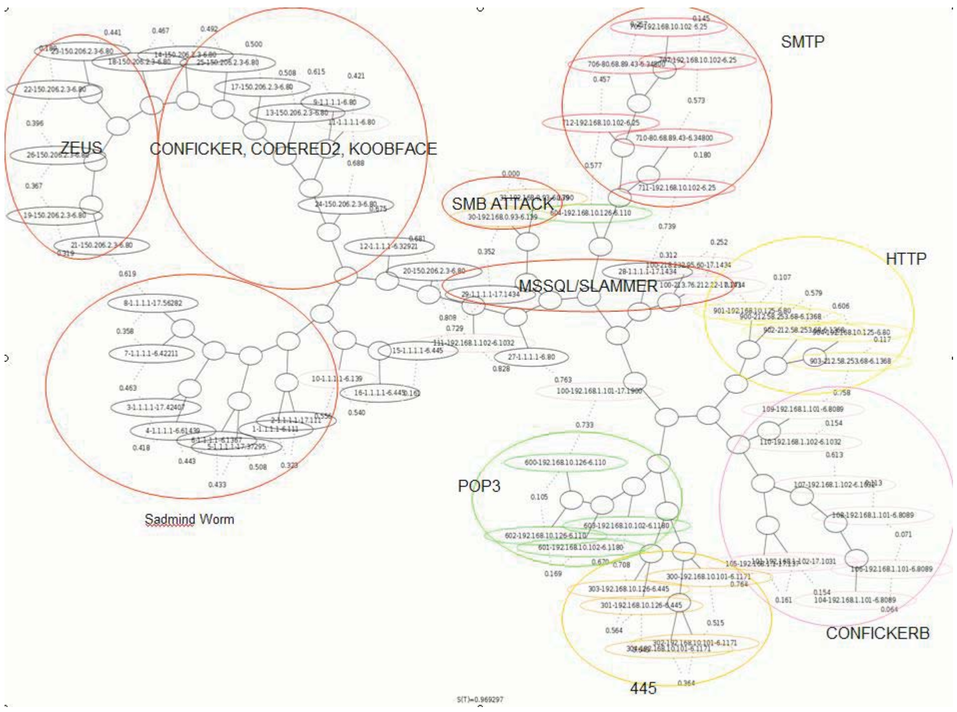


Figure 5.3: MQTC on a small dataset of mixed 100 packets of benign and worm packets. The algorithm took a long time to calculate a stable tree even for such a small dataset. The clusters are highlighted here in this figure.

Limitations: UPGMA is much faster than MQTC but slows down considerably as the dataset size increases. UPGMA took over 10 hours to process a dataset of 1000 samples.

k-Nearest Neighbour algorithm (k-NN)

k-NN is one of the simplest machine learning algorithms, which attempts to classify objects by a majority vote of its k nearest neighbours. However, determining the value of k before hand is a difficult and computationally expensive task. We tested our system with k = 1,3 and 5.

Limitations: k-NN is much faster than both MQTC and UPGMA. It was able to process 1000 samples in under 15 minutes. However, determining an optimal value for k is difficult and is prone to high false positive results.

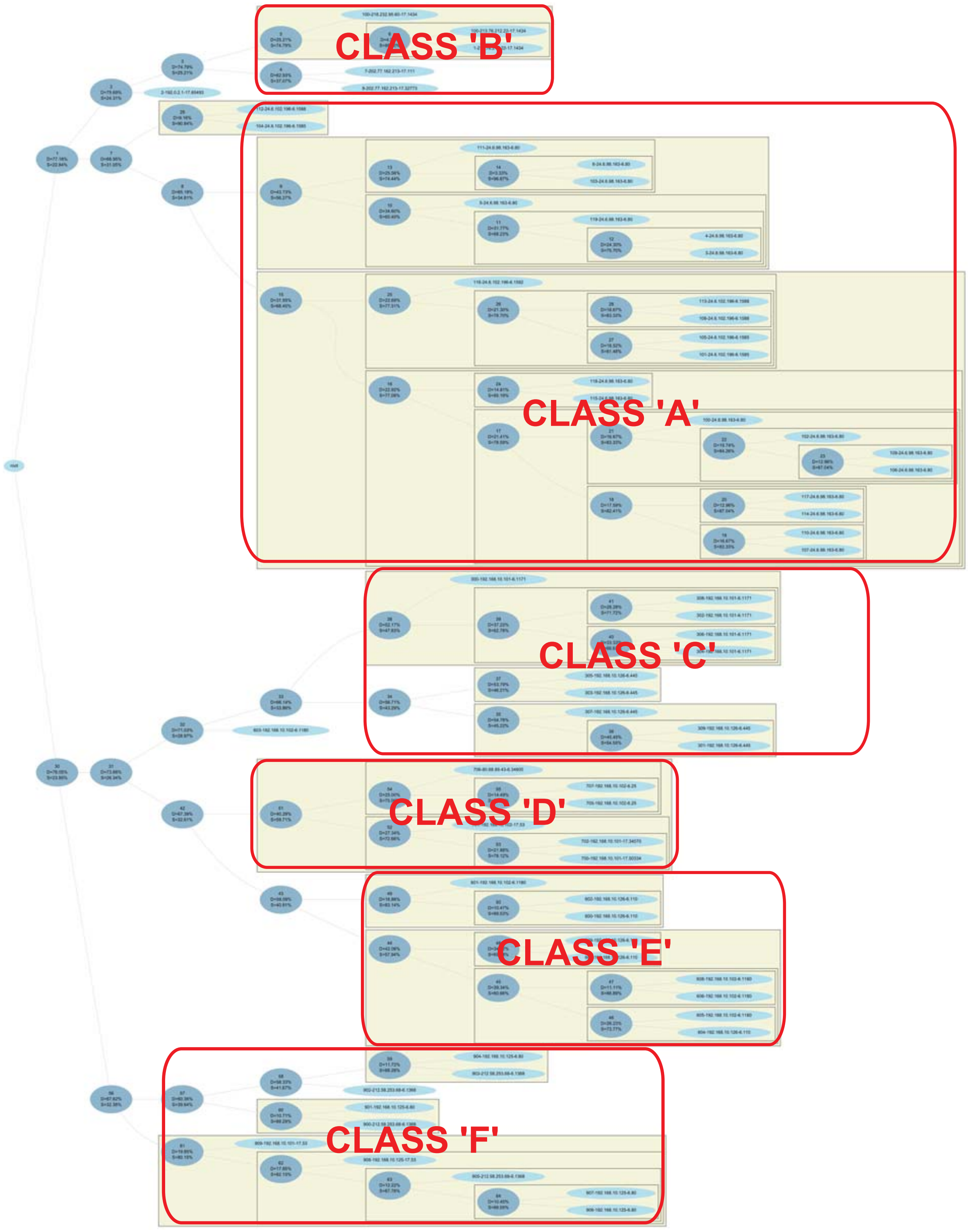


Figure 5.4: UPGMA on a small dataset @ T = 0.55. The rectangular boxes show the clusters, the dark oval nodes represent similarity and dissimilarity percentages, while the light coloured flat oval nodes represent the packets. In a very short span of time (less than a minute), UPGMA was able to correctly cluster the dataset according to its labels. UPGMA results were visualized using a modified version of Ero Carreras script (http://blog.dkbza.org/2005/07/using-complexity-theory-to-measure.html)

Threshold-based nearest neighbor or (T-NN)

T-NN is a variant of the nearest neighbour algorithm proposed for our domain. It is based on a simple linear search. For a dataset having N data points we want to find all such points in the metric space that are below the threshold value defined as a parameter at the beginning of the algorithm. These points will be considered as the nearest or closest to the reference point. For any query point, the algorithm computes distances between every other point in the dataset and adds the ones that are below the defined threshold value to the same cluster. The threshold defines the radius of the cluster. The optimal threshold value will create clusters comprising of only those points that are similar to the reference point, resulting in a perfect classification of data points from only the same class. Determination of the optimal threshold value is a challenge which directly affects the outcome of the classifier.

Limitations: Determining an optimal threshold value is a challenge.

While determining the threshold for T-NN and the value of k for k-NN might be difficult, they have shown some promising results.

5.4.5 Experiment 2: Determination of the threshold for T-NN

Central to the T-NN algorithm is a threshold value that determines the size of the classification. Determining the optimum threshold value that will maximize the true positives (TP), while minimizing false positives (FP) is a difficult task. We design two experiments in an attempt to approach this problem. In the first experiment we use a fixed threshold value, known from our past experiences to be good enough to meet the criteria, and use this to classify a given test set. This value was selected as initially a middle value between 0 and 1 that may serve as a boundary to distinguish between variants of the same class and different classes. In the second experiment we use a step-wise or adaptive threshold approach, in which the threshold value is increased in small increments of 0.05 from 0 to 1, and the accuracy is checked for each iteration. The iteration that gives the maximum accuracy is marked and the threshold for that case is considered to be the optimum threshold.

The threshold value is significant for the classifier, as it should determine the cut-off value which ascertains the classifier boundary between classes. This value should serve as input for the clustering algorithm to create clusters. One way to determine the threshold is by increasing it by small step increments of 0.05, starting from 0 and going to 1, and then calculating the accuracy of the classifier per increment in terms of true and false detection. Ideally, the threshold value having the most TP values and the least FP values for Accuracy or AUC or the one with zero FP is considered as the optimum value for that dataset.

5.4.6 Experiment 3: Determining the best Similarity Metric

To determine the optimal classification based on edit distance based similarity metrics, we used both the ROC accuracy and the ROC AUC based measures. Clustering techniques were applied to the results using an adaptive threshold to determine the clusters formed with maximum true positives (TP) and minimum false positives (FP). Next the ROC curve was plotted for result analysis. Since AUC is statistically consistent and a better discriminating measure than Accuracy (defined in the next section) (Weiss, 2004), it can be used to determine the optimal threshold value.

5.5 Results and Discussion:

5.5.1 Result 1: The best clustering algorithm for our domain

In experiment 1 presented in the previous section it was discussed in detail both theoretically and empirically using a small dataset how T-NN outperforms the other candidate clustering algorithms selected for our domain. A detailed discussing on the result using a much larger dataset comparing these clustering algorithms is given later in 5.5.6.

5.5.2 Result 2.1: T-NN with Fixed Threshold Clustering using different Similarity Metrics

Visualizations of cluster formations, for the clustering algorithms (discussed in 5.4.4) can be seen in Figures 5.5, 5.6, 5.7, and 5.8. [4]

The loosely coupled, multiple galaxies appearing in 5.5 are the clusters formed between labels of the same class measured using NCD for a threshold (T) of 0.55. The same fixed threshold value for other string metrics caused classification problems as can be seen in Figure 5.6 for the Jaro, in Figure 5.7 for the Levenshtein and in Figure 5.8 for Hamming distance. Based on these results a separate experiment (discussed in the next section) was conducted to determine the optimum threshold value for each similarity

[4]We visualized the results as undirected graphs, showing relationships between packet streams using Graphviz Ellson et al. (2002). The packet stream profiles are represented by oval nodes. The relationships are represented by a line or edges with the similarity score.

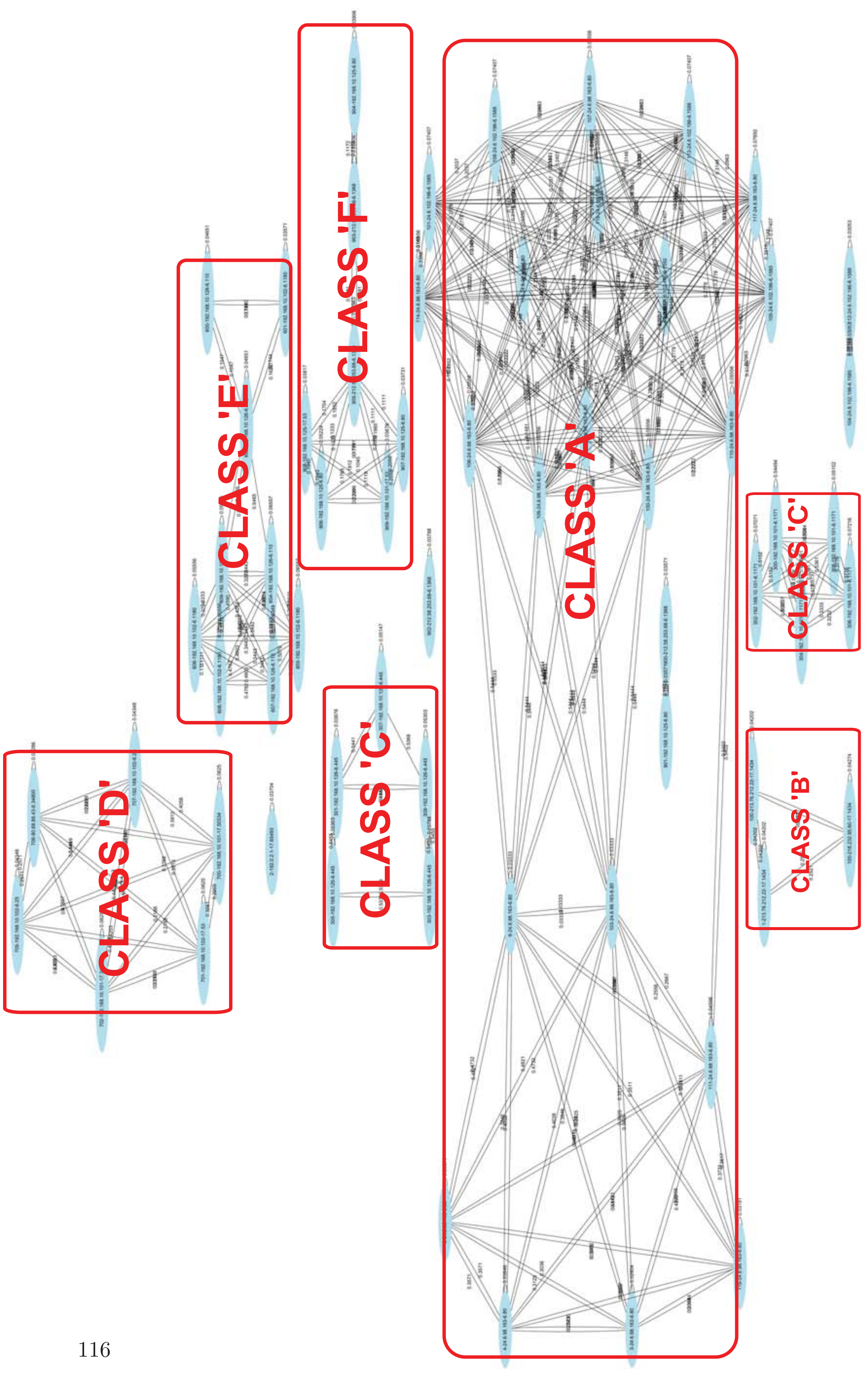


Figure 5.5: NCD on a small dataset @ T=0.55. The different galaxies represent the various clusters, the connections are represented by edges with their similarity scores. Almost all nodes seem to be part of a cluster, which shows the effectiveness of NCD @ T=0.55.

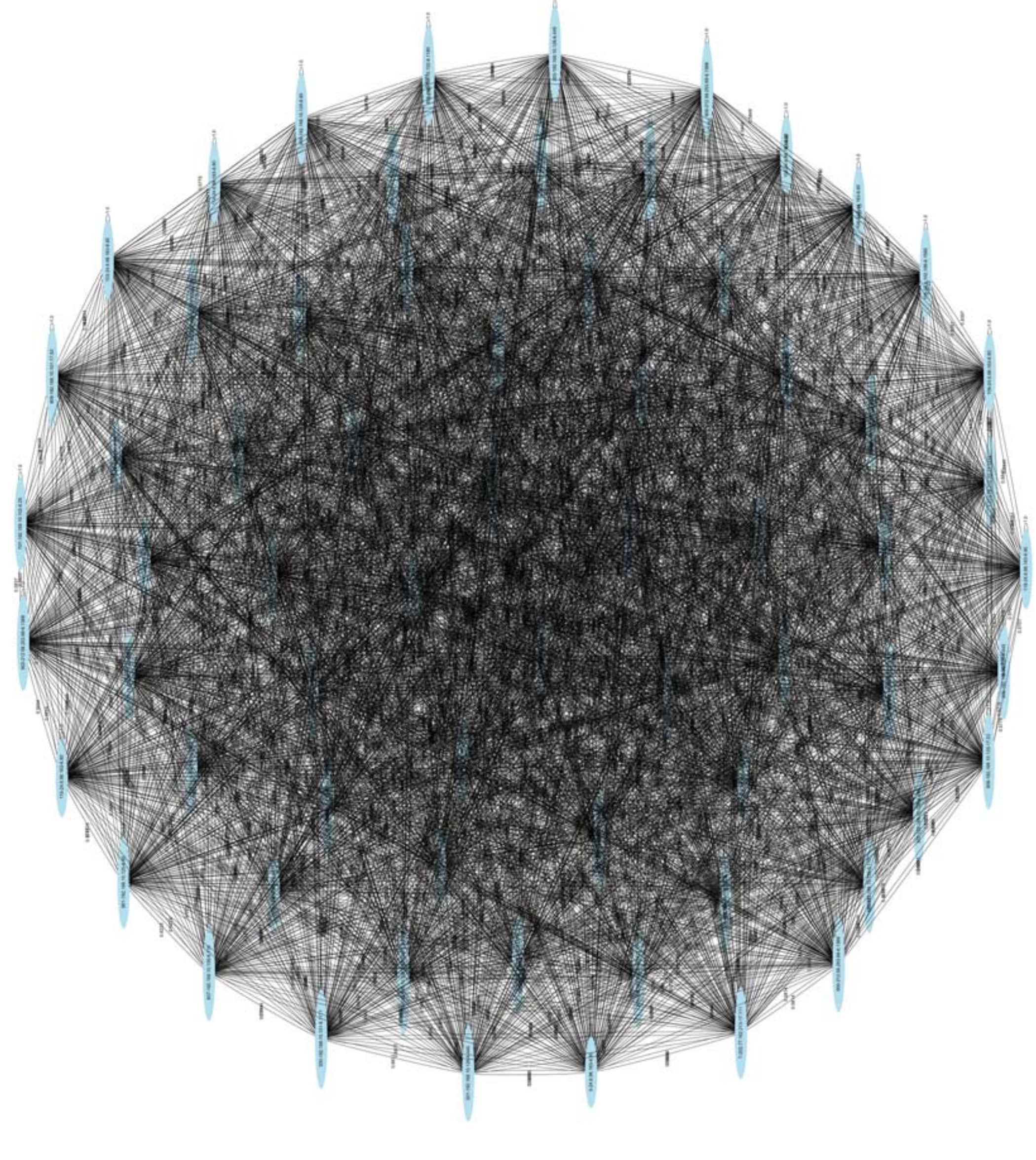


Figure 5.6: Jaro Distance calculated on a small dataset @ T=0.55 suggests that this fixed threshold is not optimal for Jaro. The big dense galaxy forming indicates misclassification at this threshold value for the Jaro algorithm. A separate experiment was conducted to determine the optimum threshold value for each similarity metric. All nodes in the dataset seem to have connections/linkages with other nodes.

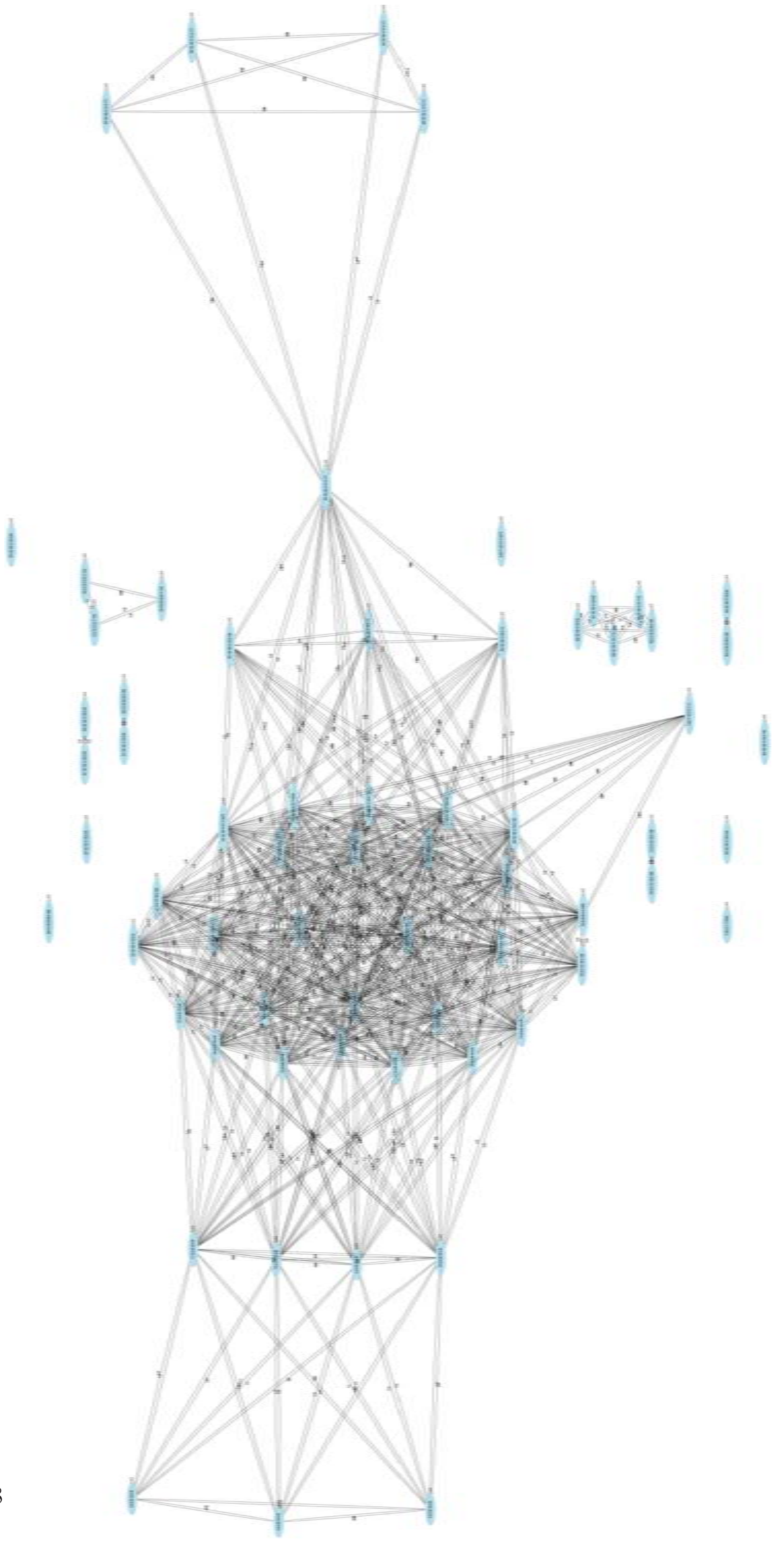


Figure 5.7: Levenshtein Distance calculated on a small dataset @ T=0.55, which is nearly half-way between 0 and 1. The results are not very good at this threshold value, as many nodes seem to be misclassified into a larger class. Very few nodes are correctly classified.

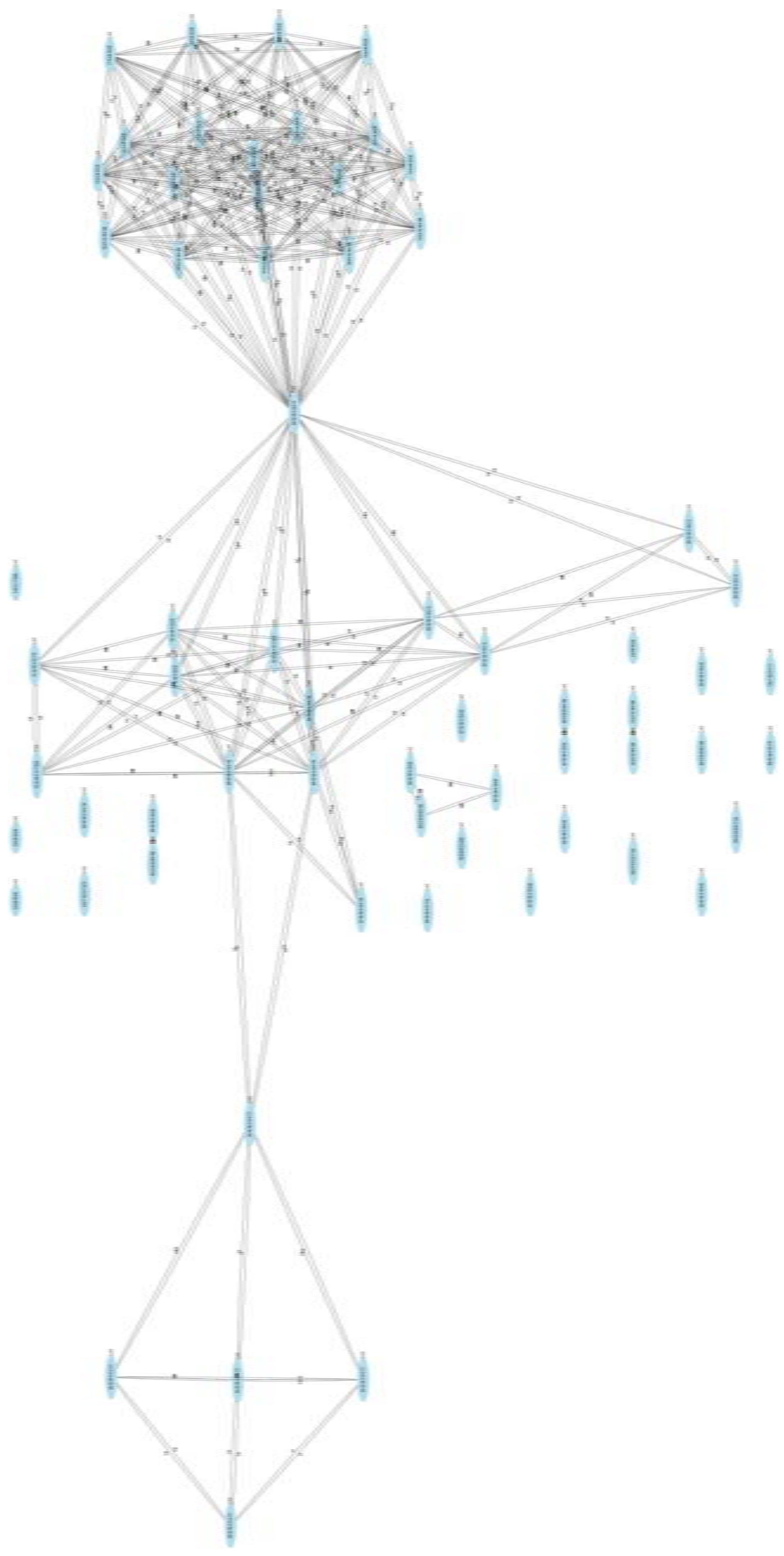


Figure 5.8: Hamming Distance on a small dataset @ T=0.55, which is nearly half-way between 0 and 1. The results are not very good at this threshold value, as many nodes seem to be misclassified into a larger class, very few nodes are correctly classified, while others are lying independently in the visualization.

metric. It can be concluded that NCD similarity metrics comparatively creates the best clusters in terms of correct classification. A partial screen-shot obtained from the NCD-based T-NN clustering on the 500 dataset is shown in Figure 5.9. This figure gives an idea of the skewed dataset. The prominent mass visible in the center is the largest class, representing the largest galaxy, while the sister galaxies around it represent other relatively smaller classes.

5.5.3 Result 2.2: T-NN Adaptive vs T-NN Fixed Threshold

From each of the 10 test sets (described in section 5.4.3), using NCD as the similarity metric, we collected accuracy results for both fixed (Discussed in section 5.4.5) and adaptive threshold (Discussed in section 5.4.5). The average accuracy expressed as a percentage was calculated for both these methods and it was observed that the fixed threshold method gave 64% accuracy compared to 87% accuracy provided by the adaptive threshold technique, as can be seen in the bar chart in Figure 5.10. This significant increase in accuracy has made the adaptive threshold technique, our technique of choice to be used with T-NN for clustering. For this experiment, the fixed threshold value was set to 0.5 and adaptive was found to vary between 0.7 - 0.8, between different test sets.

5.5.4 Threshold Results: Using T-NN with Adaptive threshold

We define the optimum threshold value as the cut-off value at which the system identifies the maximum number of True Positive (TP) values while minimizing the False Positive (FP) values. The experiments to determine the optimum threshold value are discussed in Section 5.4.5. The results in this section, followed by the discussion in Section 5.3 conclusively shows why Accuracy is not the optimal criteria for determining the threshold value, as there is a very large compromise in terms of high FP values for a small increase in the TP values. AUC provides a better compromise, with a substantial decrease in the number of FP values while maintaining a high percentage of TP.

5.5.4.1 AUC vs Accuracy vs Zero False Positive (FP).

Since AUC (when compared to Accuracy) yields a cut-off value that has fewer false positives (FP), yet maintains a high true positive (TP) value, AUC was chosen as the determining factor to identify the optimal cut-off.

The bar chart shown in Figure 5.11 shows the magnitude of the TP and FP values

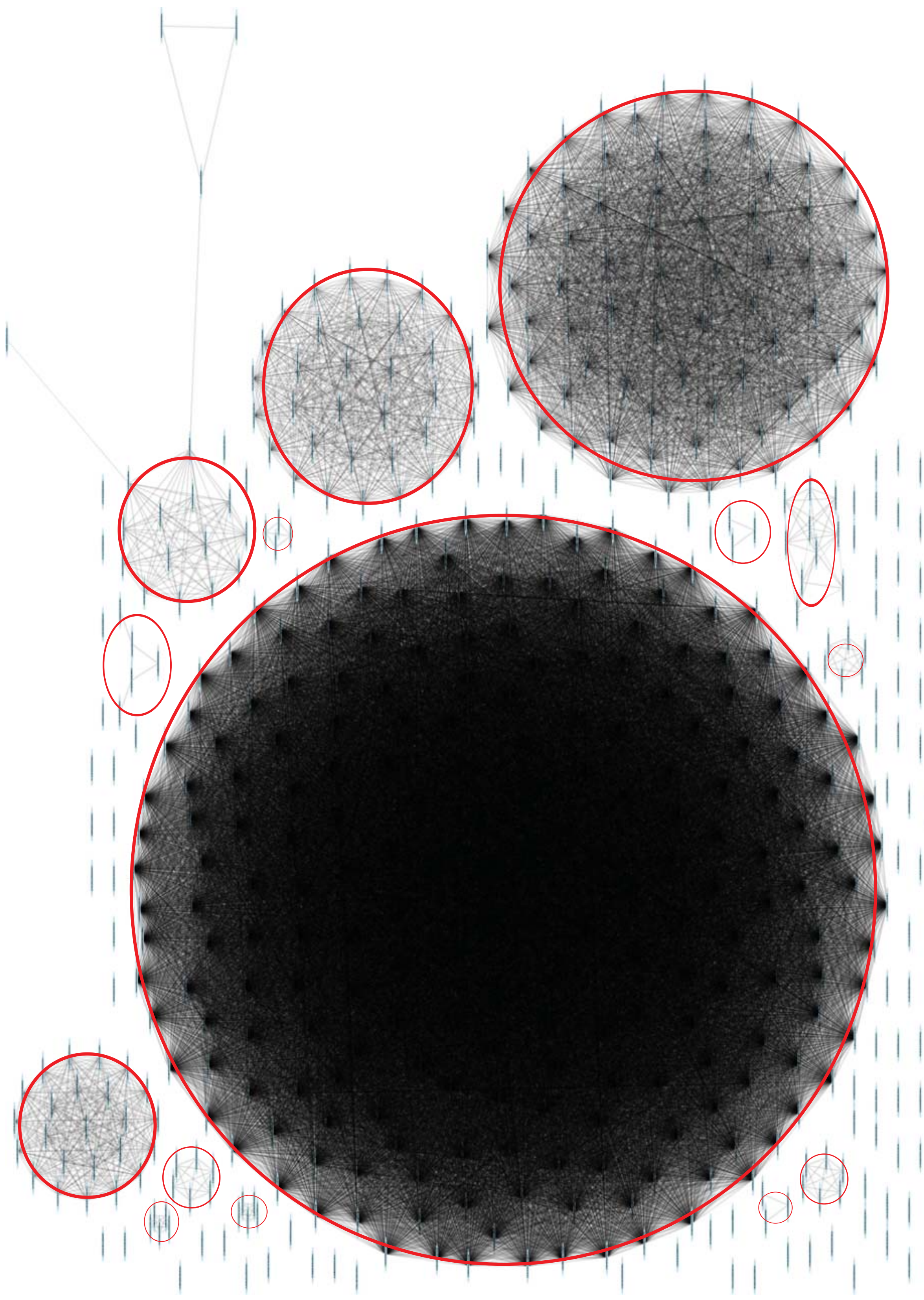


Figure 5.9: NCD on the 500 dataset @ T = 0.55. Since this is an unbalanced dataset, we see clusters of differing sizes forming. The dense dark galaxy represents the dominant class in this dataset. The subsequent smaller clusters represent the other classes as seen by the classifier. Very few nodes seem to be independent or misclassified. ROC curve analysis (discussed in 5.3) revealed that the best classification can be achieved at a threshold value of 0.75.

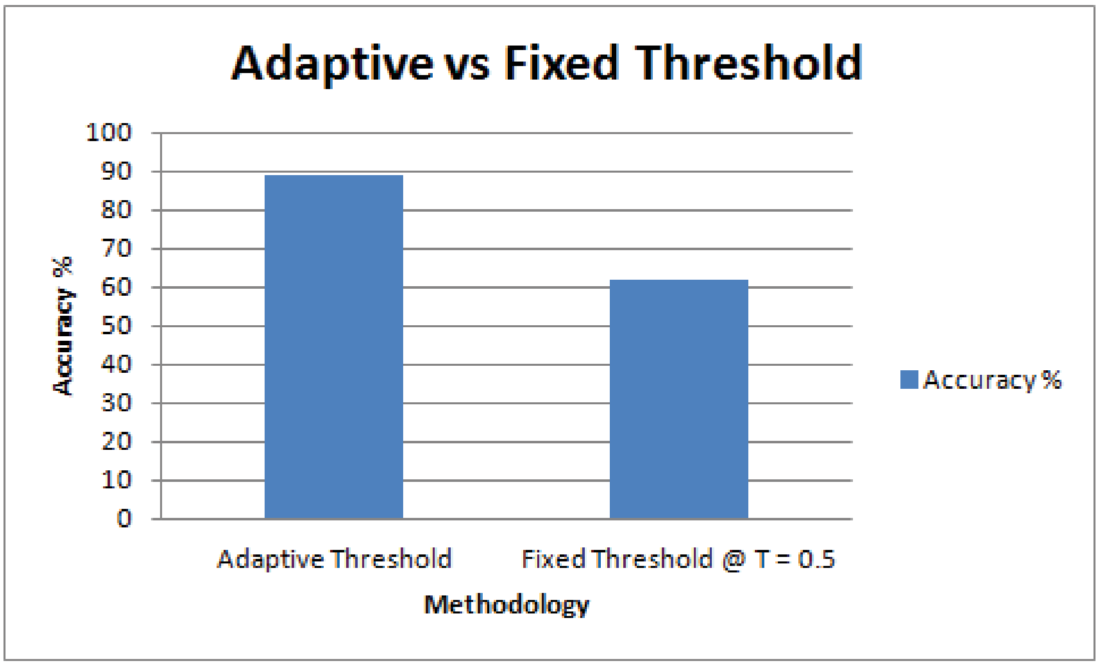


Figure 5.10: Adaptive vs fixed threshold: The adaptive threshold technique has a clear advantage over the fixed threshold technique, in terms of accuracy. Here applying an adaptive threshold gives 87% accuracy, while using a fixed threshold gives only 64% accuracy.

for AUC, Accuracy and Zero FP, across all four distance metrics. Thi graph was created using maximum value of Accuracy and AUC and maximum Accuracy at Zero FP. Ideally, a system would yield no zero false positives (FP) or false negatives (FN). However, it is evident from this figure that such a system would have a considerable reduction in the number of true positives, and a very large increase in the false negatives (FN). It can be seen from the graphs that Accuracy (approx. 88%) followed by AUC (approx. 87%) has the highest number of TP, whereas for a zero false positive system, the TP are nearly half of the Accuracy. However, the advantage is that there would be zero false positives (FP).

Figure 5.12 shows a bar chart expressing the magnitude of FP values across all four distance metrics using Accuracy, AUC and Zero FP. The graph is created using maximum value of Accuracy and AUC and maximum Accuracy at Zero FP observed from the results. Since at zero FP there would be no FP values, we shall focus on Accuracy and AUC only. Here NCD followed by Jaro is showing the lowest number of FP samples compared to the others.

NCD and Jaro for FP observations are compared in Figure 5.13. The graph created using number of FP observations resulting from the maximum value of Accuracy and AUC observed from the results. Here It can be seen that AUC has a significantly lower

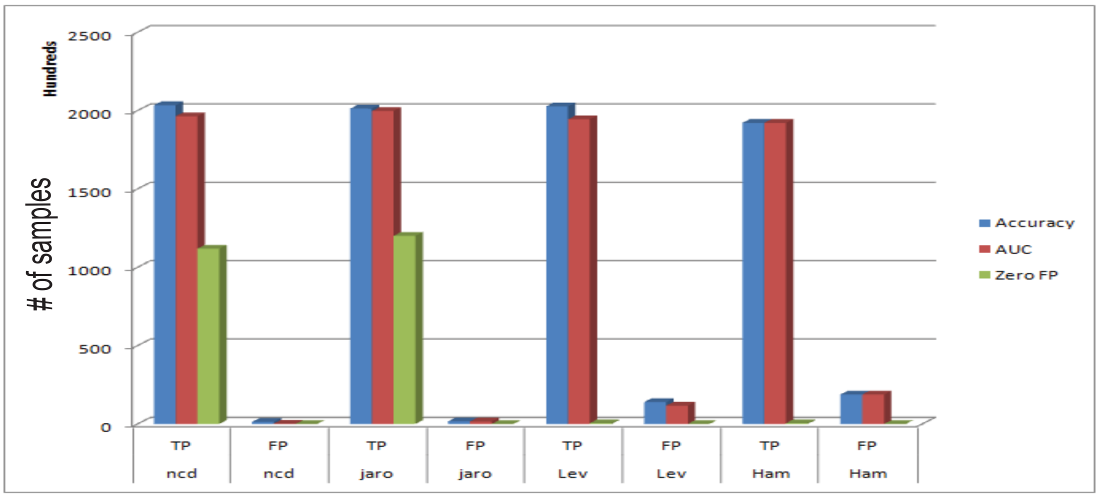


Figure 5.11: Accuracy vs AUC vs Zero FP across all metrics. This figure shows that NCD with AUC yields a high number of TP values with very few FP values compared to Accuracy. This tradeoff helps decide a suitable threshold (T) using the adaptive thresholding method for T-NN clustering.

number of FP values compared to Accuracy. Here NCD with AUC is showing the least number of FP samples, as highlighted in Figure 5.13.

5.5.5 Result 3: Best similarity metric using T-NN with adaptive threshold

We performed ROC curve analysis on all the string metrics. From the combined ROC classification results illustrated in Figure 5.14, NCD outperforms the other similarity metrics. Adding Spamsum to this comparison as shown in Figure 5.15, we observe an ROC analysis of the similarity metrics studied. It can be seen that Spamsum and NCD outperformed other similarity metrics. A high correlation ($r = 0.96$ and coefficient of determination $R^2 = 0.93$) between Spamsum and NCD can be seen visually in Figure 5.14. This confirms and endorses our observations and discussion presented earlier in the previous chapter (Similarity metrics: Hybrid approach).

5.5.6 Comparative Results: Cluster comparisons

In order to empirically compare various clustering algorithms discussed in this chapter, we selected a small subset of 1000 samples from the main dataset. We then split this dataset into a training set and a test set. The training set was created by manually picking 3 or more samples from each class. We then tested the classifier by randomly picking samples from the training set and using them as exemplars to classify the test

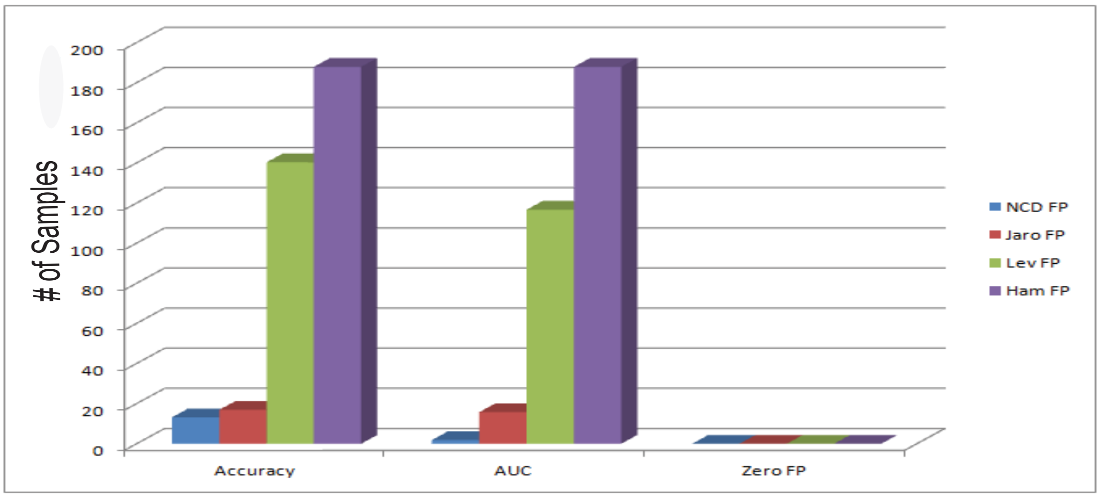


Figure 5.12: FP results for NCD, Jaro, Levenshtein and Hamming distance metrics - AUC vs Accuracy. Here NCD is showing the lowest number of FP samples compared to the others.

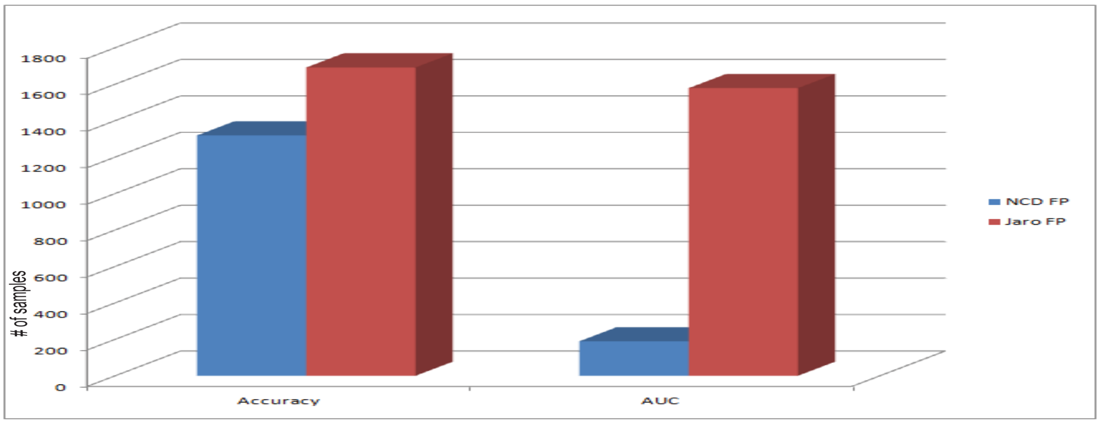


Figure 5.13: FP results for NCD and Jaro – AUC vs. Accuracy. Here NCD with AUC is showing the least number of FP samples.

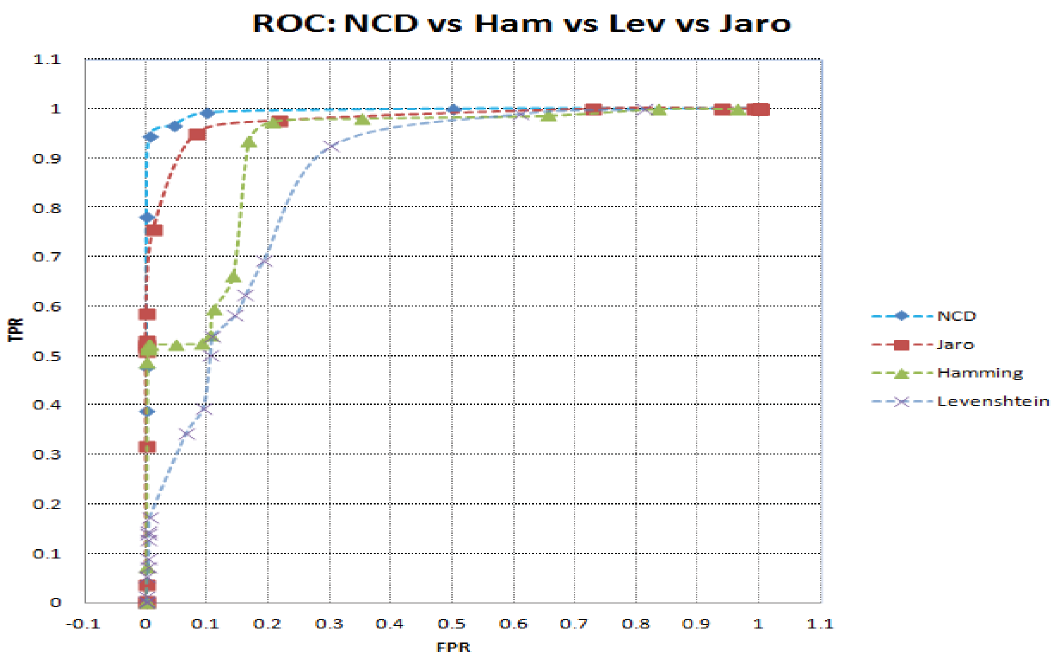


Figure 5.14: ROC: Hamming vs Levenshtein vs Jaro vs NCD, while step-wise varying the threshold. Here Hamming and Levenshtein distance performs poorly with many false positives at the start and getting worse towards the end. Jaro distance, compared to the previous two metrics performs much better, with many TP results until at the end a few FP results start to appear. As is evident from the ROC curve analysis, NCD has outperformed other similarity metrics by comparison.

set. Next we increased the number of exemplars selected from the training set and observed the accuracy of the classifier for each step. Using this methodology we compared all clustering algorithms discussed in this chapter. The result of this comparative work is illustrated in Figure 5.16. Since, for this experiment, ROC analysis will be unable to depict the number of exemplars or reference samples used as signatures, Accuracy is used on the y-axis with exemplar count or number of exemplars on the x-axis.

Discussion

As the number of exemplars or instances increase, the knowledge base of the classifier increases and hence it can perform better classification resulting in better accuracy. However, more knowledge in terms of exemplars would demand more comparisons and considerably slow down the computation process, as it undergoes a nxn or n^2 operation. Ideally, we would like our classifier to yield maximum accuracy with the lowest possible number of exemplars taken from the training set. For this particular dataset, the results illustrated in Figure 5.16 show an interesting situation where the

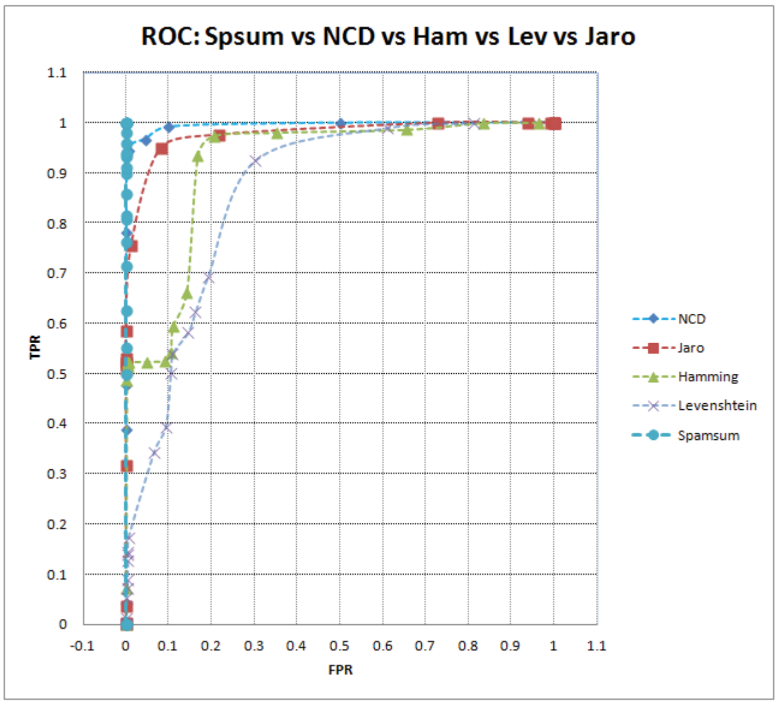


Figure 5.15: An ROC curve comparing Hamming, Levenshtein, Jaro, NCD and Spamsum similarity measures, while step-wise varying the threshold. It can be seen that Hamming and Levenshtein distance measures perform poorly, with many false positives at the start and getting worse towards the end. Jaro distance metric is much better, with true positive results until, at the end, a few false positives results start to appear. NCD and Spamsum outperform the other similarity metrics.

adaptive thresholding algorithm seems to outperform others. Step-wise or adaptive thresholding gives the best results and seems to be better than fixed thresholding and k-NN for k = 3 and 5, except when 3-NN has a very large number of exemplars (80-90). Here we can conclude that the nearest neighbour and adaptive thresholding seem to be fairly correlated. NN would select its nearest neighbour, however this sample can be a FP samples. TNN has the additional capability to reduce the search radius of NN by applying a threshold. This way only samples within the similarity threshold of an exemplar get classified by that exemplar as members of the class. Here MQTC and UPGMA could not scale due to the size of the dataset and were omitted from the comparison.

5.6 Limitations

This method has several limitations, firstly, when an attacker intentionally alters the entropy or complexity of the attack traffic or exploit to such an extent that the classifier

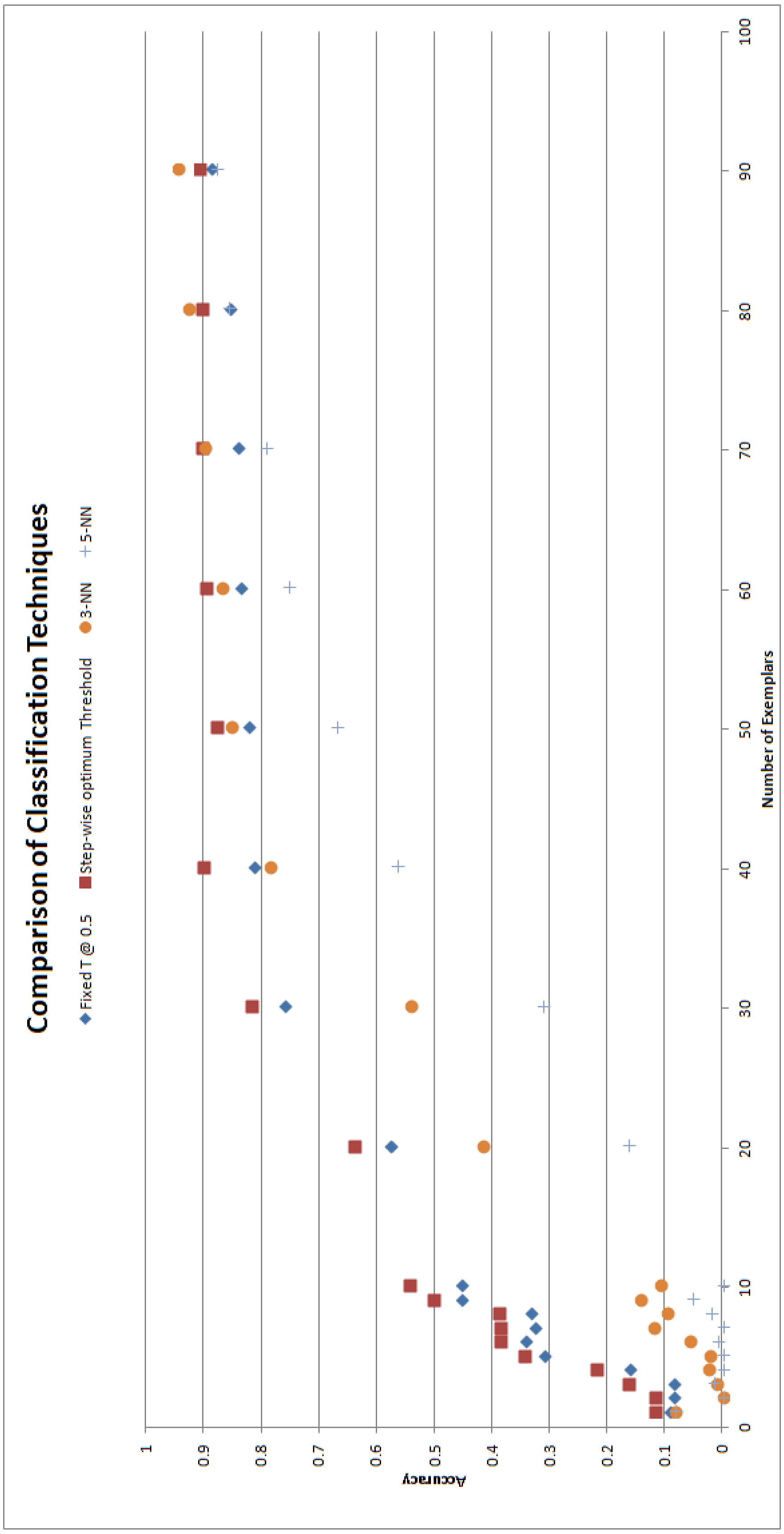


Figure 5.16: Comparative work to determine the best clustering algorithm for our domain. Adaptive thresholding performs the best, followed by fixed thresholding and k-NN for k = 3 and 5.

fails to identify it correctly. This could be achieved by adding interleaved random symbols with the actual exploit data.

Secondly, the $n \times n$ operations performed by the experiments are expensive and resource intensive in terms of both CPU and memory. It does provide a chance to study the data partitioning in more detail, but the cost to maintain such an experiment in a production environment is quite high. In order to provide efficient intrusion detection services there is a need to explore more efficient approaches, involving fewer exemplars to represent the entire dataset. Another limitation is that for both fixed and adaptive threshold methods, the same threshold value is applied across all classes. Ideally, for the adaptive threshold, the process should be extended to a per-class basis to achieve better accuracy.

Finally, we have not determined the most appropriate exemplars per-class from the training set, as currently they are being selected by a random selection function.

5.7 Conclusions

It can be concluded that the use of machine learning techniques like clustering, on results obtained from information theoretic similarity metrics, can be used for classification of malicious streams using network payload data. It was concluded that threshold plays a significant role in clustering and classification and determining an optimal threshold is a challenging task. We found that for this type of data, NCD and Spamsum metric are the most appropriate similarity metrics and concluded that ROC-AUC is a better measure than Accuracy. Finally, for evaluation we found that step-wise or adaptive thresholding (based on T-NN) is the best classification algorithm for our domain.

In the next chapter, the use of exemplar-selection algorithms to select the most important exemplars from the training set are investigated. Study involving the correlation between compression ratios and our proposed technique is conducted and comparisons between our technique with other machine learning techniques are carried out. Additionally, approaches involving automated learning of exemplar signature and threshold for optimum classification of known and novel classes are studied and compared.

Chapter 6

Exemplar Learning for Intrusion Detection

"If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle"

–*Sun Tzu, The Art of War*

In the previous chapters the most suitable clustering algorithms for our domain were identified. After a theoretical comparison, a custom algorithm was proposed. Finally all these clustering algorithms were compared with each other using a dataset. After extensive testing, it was revealed that threshold based nearest neighbour or T-NN serves as the best algorithm for clustering and classification. Later some limitations of T-NN were identified.

In this chapter we shall discuss how to utilize machine learning techniques to automatically choose the most appropriate exemplars and thresholds for each class and make our system evolve by learning exemplars from novel classes.

6.1 Introduction

Concept Learning and Exemplar Learning

Concepts are general notions that help us categorise objects, data, ideas, people or events based on some unique characteristics.

Concept learning is also referred to as category learning, concept attainment and concept formation. It is defined as: "the search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories"[1].

In other words, concept learning is a learning method that requires a machine learner to be able to compare, distinguish and classify objects based on unique features or characteristics extracted from examples, it has been trained on. General concepts learned from prior examples are used to classify new examples.

Exemplar learning involves the learner choosing exemplars or instances from a set of objects based on their resemblance to a certain category or group. The identification of an object as an exemplar or instance and the rules to determine matching objects group membership are the two key measurements that the exemplar theory depends on.

In machine learning terms, concept learning comes under the category of supervised learning. A class labelled dataset is provided to the algorithm to train on. The algorithm or learner uses exemplars from this labelled training data set to create generalizations to classify other objects. Such algorithms are often referred to as instance learning algorithms or lazy learning algorithms.

6.2 Problem Description

Given the data and the similarity metric, there are four problems for the exemplar learner:

1. To identify and select suitable exemplars or instances from the training set (TR), (Instance Selection Problem).

2. To learn an appropriate similarity threshold, to determine data points which belong to the same class as the exemplar class, for better classification.

[1]Defined by cognitive psychologist Jerome Bruner. Bruner, Goodnow, and Austin (1967)

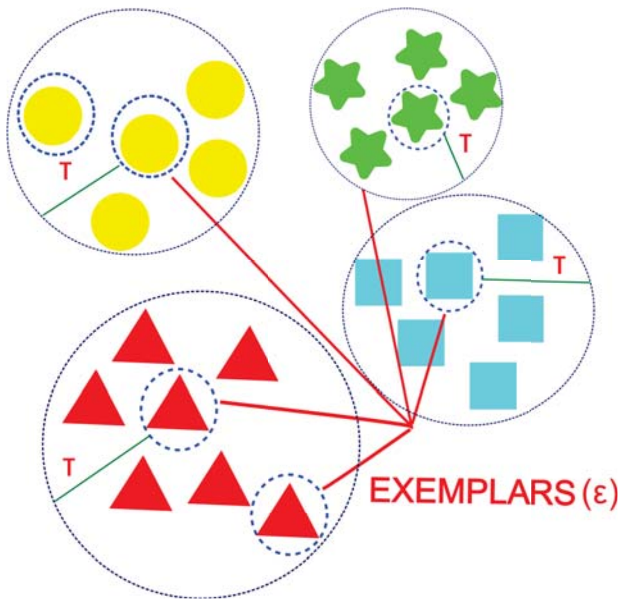


Figure 6.1: For categories a, b, c, d, each with a set of samples such as $a_1 \ldots a_n$, a model can be defined as a list of tuples, where each tuple comprises of an exemplar a_ϵ selected from the samples and a corresponding distance threshold value a_τ. If there are significant variations within one class, there may be several exemplars from the same class.

3. To identify unknown or novel samples that did not get classified by prior knowledge extracted from the training set (TR).

4. To select appropriate exemplars from these unknown or novel samples for classification.

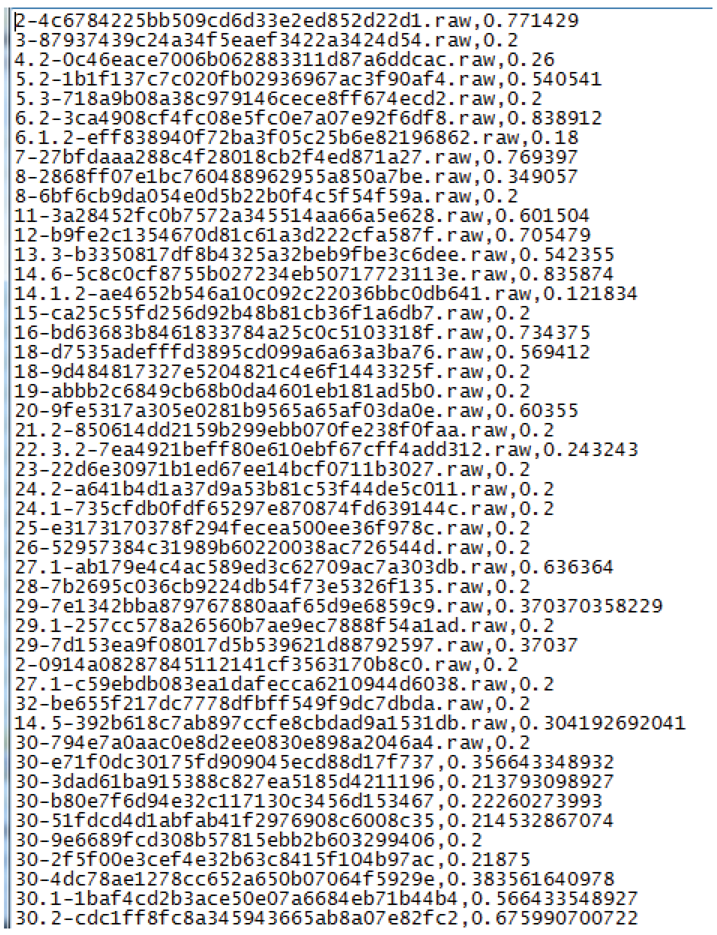
6.3 Methodology and Algorithm Design

6.3.1 Exemplar Learning for Intrusion Detection

We have chosen to use exemplar learning, since we have a labelled dataset of previously known attacks available, and since at a later stage we expect that novel attacks, which will be detected automatically by our system, will be analysed by humans (or eventually other software systems) in order to name and identify suitable responses to them.

After selection of the appropriate exemplars and their corresponding thresholds, the acquired model consists of a set of (exemplar data point, threshold) tuples. A schematic of this is shown in figure 6.1 and figure 6.2.

Choosing the exemplars appropriately is important. A poorly selected exemplar may result in:



```
2-4c6784225bb509cd6d33e2ed852d22d1.raw,0.771429
3-87937439c24a34f5eaef3422a3424d54.raw,0.2
4.2-0c46eace7006b062883311d87a6ddcac.raw,0.26
5.2-1b1f137c7c020fb02936967ac3f90af4.raw,0.540541
5.3-718a9b08a38c979146cece8ff674ecd2.raw,0.2
6.2-3ca4908cf4fc08e5fc0e7a07e92f6df8.raw,0.838912
6.1.2-eff838940f72ba3f05c25b6e82196862.raw,0.18
7-27bfdaaa288c4f28018cb2f4ed871a27.raw,0.769397
8-2868ff07e1bc760488962955a850a7be.raw,0.349057
8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw,0.2
11-3a28452fc0b7572a345514aa66a5e628.raw,0.601504
12-b9fe2c1354670d81c61a3d222cfa587f.raw,0.705479
13.3-b3350817df8b4325a32beb9fbe3c6dee.raw,0.542355
14.6-5c8c0cf8755b027234eb50717723113e.raw,0.835874
14.1.2-ae4652b546a10c092c22036bbc0db641.raw,0.121834
15-ca25c55fd256d92b48b81cb36f1a6db7.raw,0.2
16-bd63683b8461833784a25c0c5103318f.raw,0.734375
18-d7535adefffd3895cd099a6a63a3ba76.raw,0.569412
18-9d484817327e5204821c4e6f1443325f.raw,0.2
19-abbb2c6849cb68b0da4601eb181ad5b0.raw,0.2
20-9fe5317a305e0281b9565a65af03da0e.raw,0.60355
21.2-850614dd2159b299ebb070fe238f0faa.raw,0.2
22.3.2-7ea4921beff80e610ebf67cff4add312.raw,0.243243
23-22d6e30971b1ed67ee14bcf0711b3027.raw,0.2
24.2-a641b4d1a37d9a53b81c53f44de5c011.raw,0.2
24.1-735cfdb0fdf65297e870874fd639144c.raw,0.2
25-e3173170378f294fecea500ee36f978c.raw,0.2
26-52957384c31989b60220038ac726544d.raw,0.2
27.1-ab179e4c4ac589ed3c62709ac7a303db.raw,0.636364
28-7b2695c036cb9224db54f73e5326f135.raw,0.2
29-7e1342bba879767880aaf65d9e6859c9.raw,0.37037035822
29.1-257cc578a26560b7ae9ec7888f54a1ad.raw,0.2
29-7d153ea9f08017d5b539621d88792597.raw,0.37037
2-0914a08287845112141cf3563170b8c0.raw,0.2
27.1-c59ebdb083ea1dafecca6210944d6038.raw,0.2
32-be655f217dc7778dfbff549f9dc7dbda.raw,0.2
14.5-392b618c7ab897ccfe8cbdad9a1531db.raw,0.304192692041
30-794e7a0aac0e8d2ee0830e898a2046a4.raw,0.2
30-e71f0dc30175fd909045ecd88d17f737,0.356643348932
30-3dad61ba915388c827ea5185d4211196,0.213793098927
30-b80e7f6d94e32c117130c3456d153467,0.22260273993
30-51fdcd4d1abfab41f2976908c6008c35,0.214532867074
30-9e6689fcd308b57815ebb2b603299406,0.2
30-2f5f00e3cef4e32b63c8415f104b97ac,0.21875
30-4dc78ae1278cc652a650b07064f5929e,0.383561640978
30.1-1baf4cd2b3ace50e07a6684eb71b44b4,0.566433548927
30.2-cdc1ff8fc8a345943665ab8a07e82fc2,0.675990700722
```

Figure 6.2: The model written out to a csv file. Each row can be read as: The labelled exemplar ϵ , and the threshold τ. There can be several exemplars from the same class (e.g. class 30 in this case has several exemplars), indicating rich variations within the class samples or candidate sub-classes. Otherwise if all samples in the class are quite similar, then they can be represented by a single exemplar.

1. a large number of false positives: Mainly due to exemplars selected from the class boundaries, which may exhibit more similarity with samples of other classes than their parent class and therefore result in high FP by classifying samples from a different or negative class.

2. a large number of exemplars being required: If a few bad exemplars, which do not show similarity with other class members are selected to create a model, then they could degrade the classifiers accuracy. This will result in more exemplars randomly being selected from the training set (TR) to increase the classifiers accuracy. This would be detrimental to the classifier as it would result in a very large and bulky model and would require that many more data points to process for classification.

This is particularly important when the training dataset is unbalanced or skewed (due to class imbalance), when there are a very small number of samples of some classes (minority class), while other classes might have more samples (majority class).

6.3.2 Model Creation using exemplar-based learning

In machine learning, the performance of a learning classifier depends directly on the effectiveness of its mathematical or statistical model. In other words, how well the given dataset or algorithm output fits to the prescribed model. Model selection requires selecting the best or most appropriate model from candidate models, to solve a given machine learning problem.

We have an imbalanced labelled dataset of malicious network streams that we want to classify. An imbalanced dataset resembles real traffic. For classification or supervised learning problems the classifier is trained on classified examples and is expected to classify unseen samples. It is assumed that every instance (input sample) belongs to one, and only one, class. The model M would include multiple instances of exemplar ϵ and threshold τ tuples. Our model will be able to determine the best exemplar(s) per category, and suggest the optimal threshold for that exemplar, for classification. The design of this exemplar-based learning classifier model is now discussed.

6.3.2.1 Algorithm 1: Exemplar Selection (Choosing Exemplars from TR)

Choosing an exemplar for each category is a difficult task. A poorly selected exemplar may result in a single, noisy exemplar, which in turn may result in a large number of false positives. This may be due to the imbalanced/skewed dataset or the nature of the samples in it. It is required to prune the noisy exemplars (Edition(Aha et al., 1991) or ENN discussed in section 6.5.4) and update the model with the appropriate exemplars and threshold values per class (Condensation(Aha et al., 1991) or CNN discussed in section 6.5.4).

Algorithm 1 shows our exemplar selection algorithm. As a first step, an $n \times n$ similarity matrix of samples in the training set (TR) is calculated. From this matrix, an exemplar ϵ for a category is chosen by comparing all samples in a particular class to find the one with the lowest overall distance (i.e. greatest similarity). This is done by taking the minimum of the sum of all corresponding similarity scores of the data points for each sample. This sample then becomes our exemplar label.

The threshold value is then calculated by taking the max of the corresponding similarity scores of this sample with others in the same category. The optimal threshold value is determined by testing for false-positives on each data point below the initial threshold, until a value with no false-positives is established. Note that this assumes that each data point belongs to precisely one class, which is true for the datasets that we have analysed. The algorithm iterates over the remaining points for that category, until all the points are accounted for by the model. Finally a sanity check is performed on the model to ensure that that the threshold τ satisfies the condition $0.1 < \tau < 0.8$. Here a very tight threshold i.e. less than or equal to 0.1 will encompass very few or data points that are exactly the same as the exemplar. This inhibits the ability of the system to be able to detect variations in the form of similar samples and not just the same samples. A very high threshold i.e. greater than 0.8 would cause false-positives due to negative class classification.

Algorithm 1 Calculate $model = train$(labelled distance matrix, list of classes)

```
Require: list_of_class , similarity_matrix
Ensure: list_of_outliers
 1: for each class C in list_of_class do
 2:     find the datapoint $p \in C$ that has minimum distance to all points in $C$
 3:     set initial threshold $\tau$ to max (distance from $p$ to all elements of $C$)
 4:     if any exemplar is within threshold $\tau$ distance of $p$ then
 5:         set threshold $\tau$ = first point below min(distance from $p$ to exemplars)
 6:         check that $0.1 < \tau < 0.8$, otherwise set to bottom/top of range
 7:     end if
 8:     $add(p, \tau)$ to model
 9:     remove all points within distance $\tau$ of $p$
10:     if any points in $C$ remain then
11:         leave $C$ in list_of_classes
12:     else
13:         remove C from list_of_classes
14:     end if
15: end for
```

6.3.3 Algorithm 2: Lifelong Learning (Detecting and adding novel input classes)

The work done for Algorithm 1 involved creating a model (comprising of exemplars and thresholds) from a known labelled training dataset and then trying to classify a large test/target dataset with this model. This work was very much a two-class classification problem, where all the samples in the set were classified as one of the classes identified in the model.

There were, however, a few unknown samples which did not get classified/identified

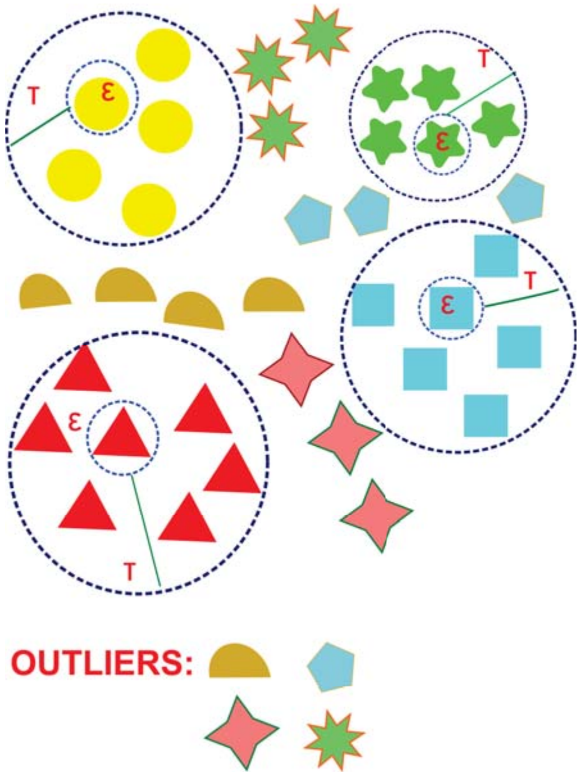


Figure 6.3: Novelty detection

by any of the classes in the model that were scattered across between the boundaries of known classes in multidimensional space. Identification and classification of these unknown or novel data samples became a motivation to study and solve the problem of novelty detection for our domain. In the literature, the terms One-class classification, novelty detection, outlier detection and concept learning refer to different applications of the same problem.

Using the algorithm given in the previous section, we are able to identify a set of exemplars and their corresponding distance thresholds. New input data can now be classified into one of the existing classes by simply comparing it to each exemplar in order and checking if it is within the appropriate threshold distance. However, we also want to deal with novel exemplars that are introduced and do not match any of the current classes in the model. All such samples are added to a set of outliers. A pictorial manifestation of this is shown in Figure 6.3.

Algorithm 2 describes a methodology to:

1. Classify the outliers into new classes

2. Select exemplars and thresholds for these new classes

3. Add these new exemplars and thresholds to the model to enhance classification.

Any data point not recognised as belonging to any class is added to a list of outliers. The algorithm first calculates the distance between the exemplars ϵ_{model} from the model and all the points in the list of outliers. This results in a $n \times m$ distance matrix of similarity scores. An initial outlier exemplar $\epsilon_{outlier}$ is chosen as the data point p in the list of outliers with the lowest score to all the other outlier data points and the current exemplars in the model ϵ_{model}. The initial threshold $\tau_{outlier}$ is set to the maximum score observed from the corresponding scores for p. Next, the exemplar $\epsilon_{outlier}$ and threshold $\tau_{outlier}$ are tested for false positives.

For the novelty detection algorithm, the false positive observations would be the exemplar ϵ_{model} data points taken from the model. If any exemplar ϵ_{model} from the input model has a similarity below the initial threshold $\tau_{outlier}$ of p, it is considered to be a false positive, while the true positive values would be all data points in the list of outliers, under the test threshold. In the case where false positives exist, the threshold is adjusted to the maximum true positive value that is less than the minimum cost false positive observation. A sanity check is performed to ensure that the threshold $\tau_{outlier}$ value satisfies the condition $0.1 < \tau < 0.8$.

This new exemplar p or $\epsilon_{outlier}$ and its adjusted threshold $\tau_{outlier'}$ is then appended to a list of exemplars in the model and this exemplar and the true positive data points that it was able to classify are removed from the list of outliers. The process is repeated until there are no more items remaining in the list of outliers.

Algorithm 2 Calculate $update\_model = add\_Exemplar(list\_of\_outliers, similarity\_score\_matrix, model)$

```
Require: list_of_class , similarity_matrix, list_of_outliers
Ensure:
1: while outliers in list_of_outliers do
2:     find the data point $p$ that has minimum distance to all points in similarity_score_matrix
3:     set initial threshold $\tau$ to max (distance from $p$ to all outliers)
4:     if any exemplar is within threshold $\tau$ distance of $p$ then
5:         set threshold $\tau$ = first point below min(distance from $p$ to exemplars)
6:         check that $0.1 < \tau < 0.8$, otherwise set to bottom/top of range
7:     end if
8:     remove all outliers within distance $\tau$ of $p$
9:     $add(p, \tau)$ to model
10: end while
```

6.4 Experiments and Validation

To validate this approach, a total of 6 experiments were conducted:

1. Experiment 1: Exemplar Selection (Supervised) 6.4.1.1
2. Experiment 2: Novelty Detection (Supervised) 6.4.2.1
3. Experiment 3: Novelty Detection (Unsupervised) 6.4.2.3
4. Experiment 4: Instance Selection 6.5.1
5. Experiment 5: Threshold Selection 6.5.2
6. Experiment 6: Comparison with other Instance Selection Algorithms (ENN) 6.5.4.2

6.4.1 Validation of the Exemplar Selection Algorithm

6.4.1.1 Experiment 1: Exemplar Selection (Supervised)

In order to validate Algorithm 1 we conducted an experiment in which we randomly selected samples from the training set (TR) to create a model and compare the results with the model provided by this algorithm. A key question here would be that: how do we determine a threshold value for the exemplar? Do we need to run a sensitivity test for that? Such a test will involve sliding the threshold in small increments to determine the threshold value that will encompass maximum classification. This will be time consuming and far less efficient, as randomly selecting exemplars can never guarantee good classification by the classifier, as the classes may contain sub-classes, which might not get classified at all by the randomly chosen exemplar.

As an initial, 'straw-man' test we created a training set made up of of 300 samples, representing 30 known classes, and a test set comprising of 1200 samples from the same 30 classes. Both the training set and the test set were selected from the dataset described in Chapter 3. We used a set of exemplars chosen at random from the training set, up to three for each class, and assigned each of them a fixed threshold value of 0.5 (this is similar to the T-NN Fixed method discussed in the previous chapter). We did this for 10 different random choices of exemplar sets, and saw an overall average accuracy of around 70%. We observed that a large part of the problem with this method is that 0.5 is not a fair threshold to choose, and so we repeated the experiment, but this time determined the threshold by starting from 0.3 and increasing it in small

Method	Exemplars	% Accuracy
TNN-Fixed	120	70
TNN-Adaptive	120	88
TNN using Algorithm 1	49	95

Table 6.1: Comparison of methods random exemplars vs selected exemplars

increments of 0.1 to determine the maximum classification (this is similar to the T-NN Adaptive method discussed in previous chapter). This is time-consuming, but makes a fair test.

6.4.1.2 Results of Experiment 1

We performed 10 iterations of Experiment 1 and found an average accuracy of 88%. In contrast, using our algorithm to select exemplars and thresholds led to 95% accuracy, and used only 49 exemplars instead of the 120 used by the random method.

We found that the experiments based on randomly selecting exemplars led to fluctuating results. The highest accuracy results obtained via randomly picking exemplars led to an overall accuracy of 92%, compared to 95% using our algorithm. This is a significant increase in the classifier's performance and provides an automated and robust method of exemplar selection for model creation. The relatively high average percentage (88%) in terms of accuracy for the random exemplar-selection methodology is due to the fact that we were picking five exemplars per class and optimising the choice of threshold for each, which is considerably more computationally expensive than using our algorithm. The results are summarized in table 6.1. It also points to the fact that similarity within a category of network streams exists because of the protocol design, which reduces complexity and enhances the similarity for classification.

The same experiment was repeated on a larger dataset of 6600 samples and similar results were obtained. A graphical visualization of the results can be seen in Figure 6.4, 6.5, 6.6 and 6.7.

6.4.1.3 Discussion

Compared to T-NN with adaptive thresholding, our algorithm requires fewer exemplars with more definitive thresholds. On the 300 sample training set, our algorithm proposed only 49 exemplars compared to 120 exemplars. Hence with only one-third the number of exemplars and with a more accurate or definitive threshold in the model we get at

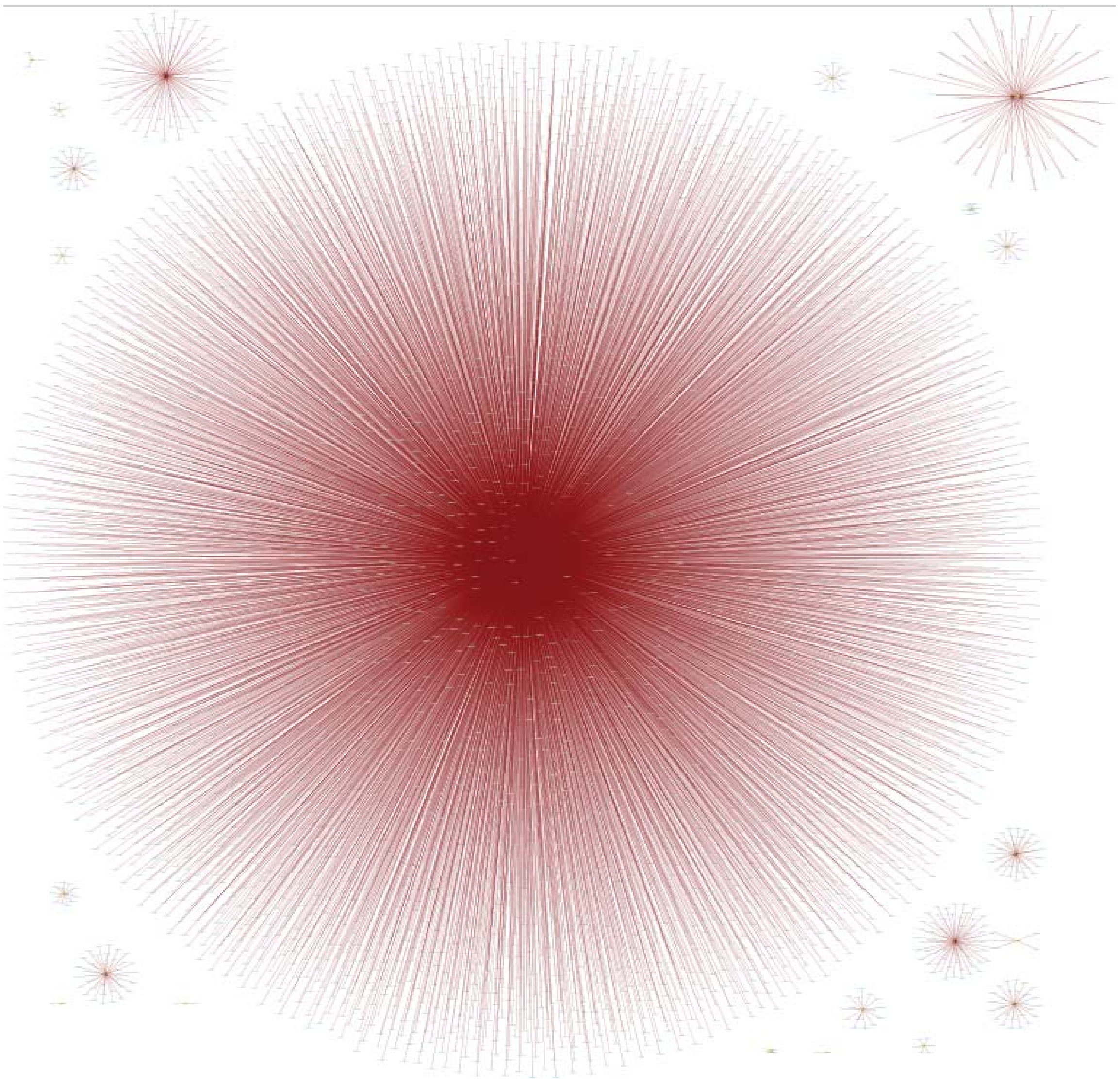


Figure 6.4: An aerial view of the classification. We can see multiple clusters forming, with one large cluster in the center representing the majority class. While the minority classes are shown in the sides. This graphic gives a clear picture and idea of the different sizes of classes. We used 49 exemplars to successfully cluster this test set .

least a 10% increase in terms of accuracy.

6.4.2 Validation of the Novelty Detection Algorithm

The validation of the novelty detection algorithm was motivated by the following two questions:

1. Can the novelty detection algorithm detect known classes, which were removed

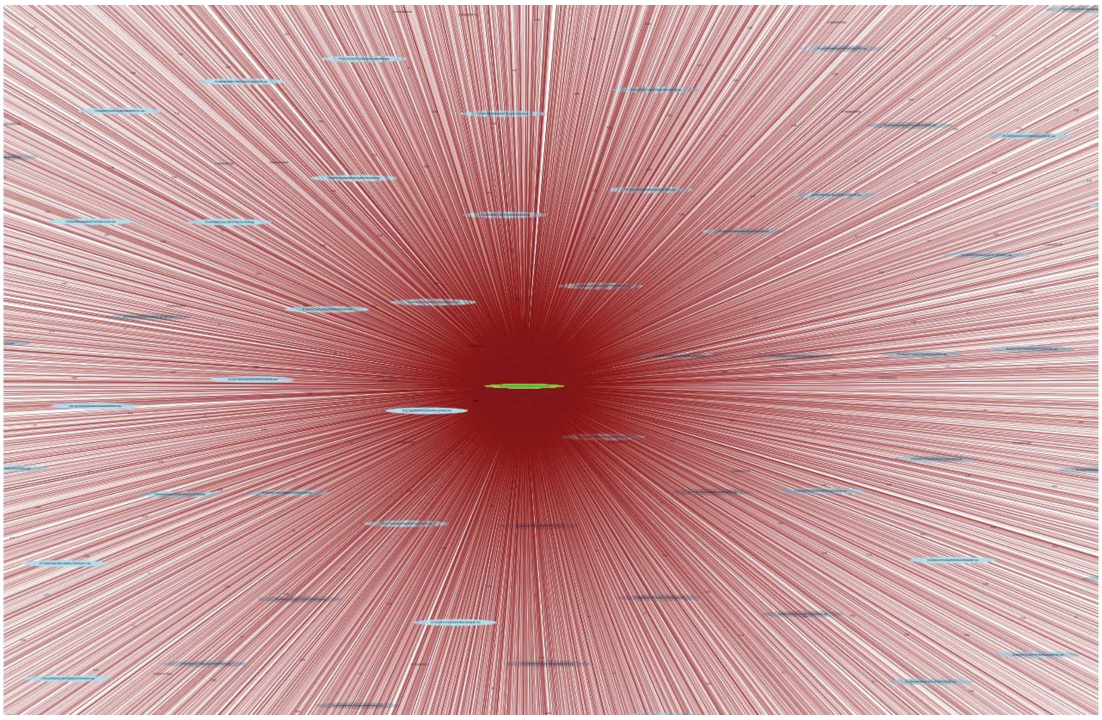


Figure 6.5: The core of the large cluster is visible here. It has 1 exemplar, represented as green node and multiple samples of the same class, represented as pale blue nodes.

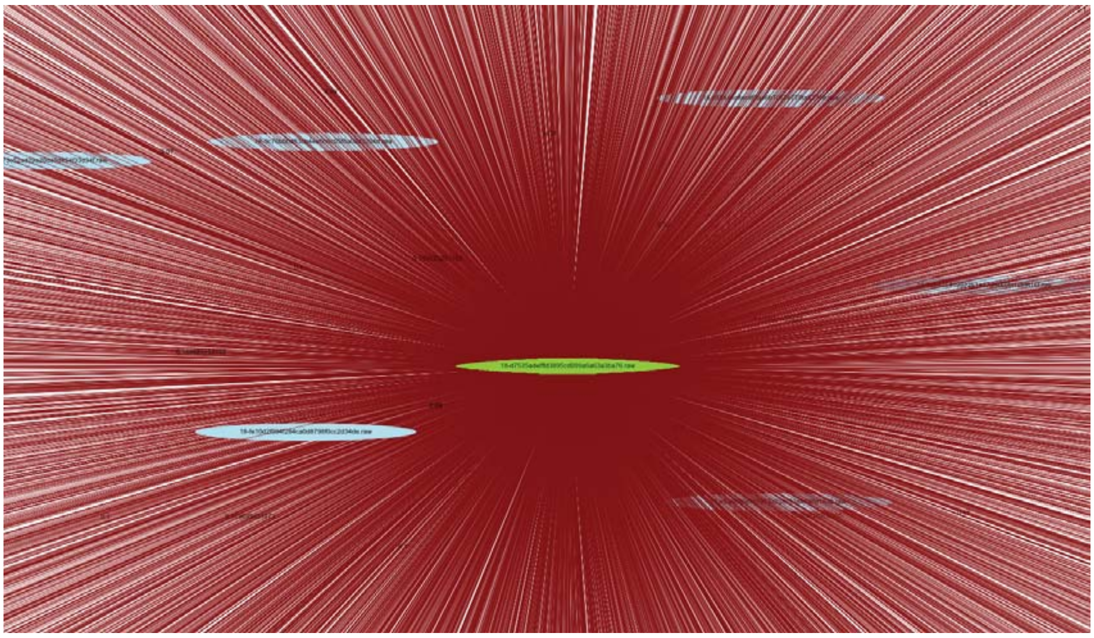


Figure 6.6: Enlarging previous figure: The core of the large cluster is visible here. It has 1 exemplar, represented as green node and multiple samples of the same class, represented as blue nodes.

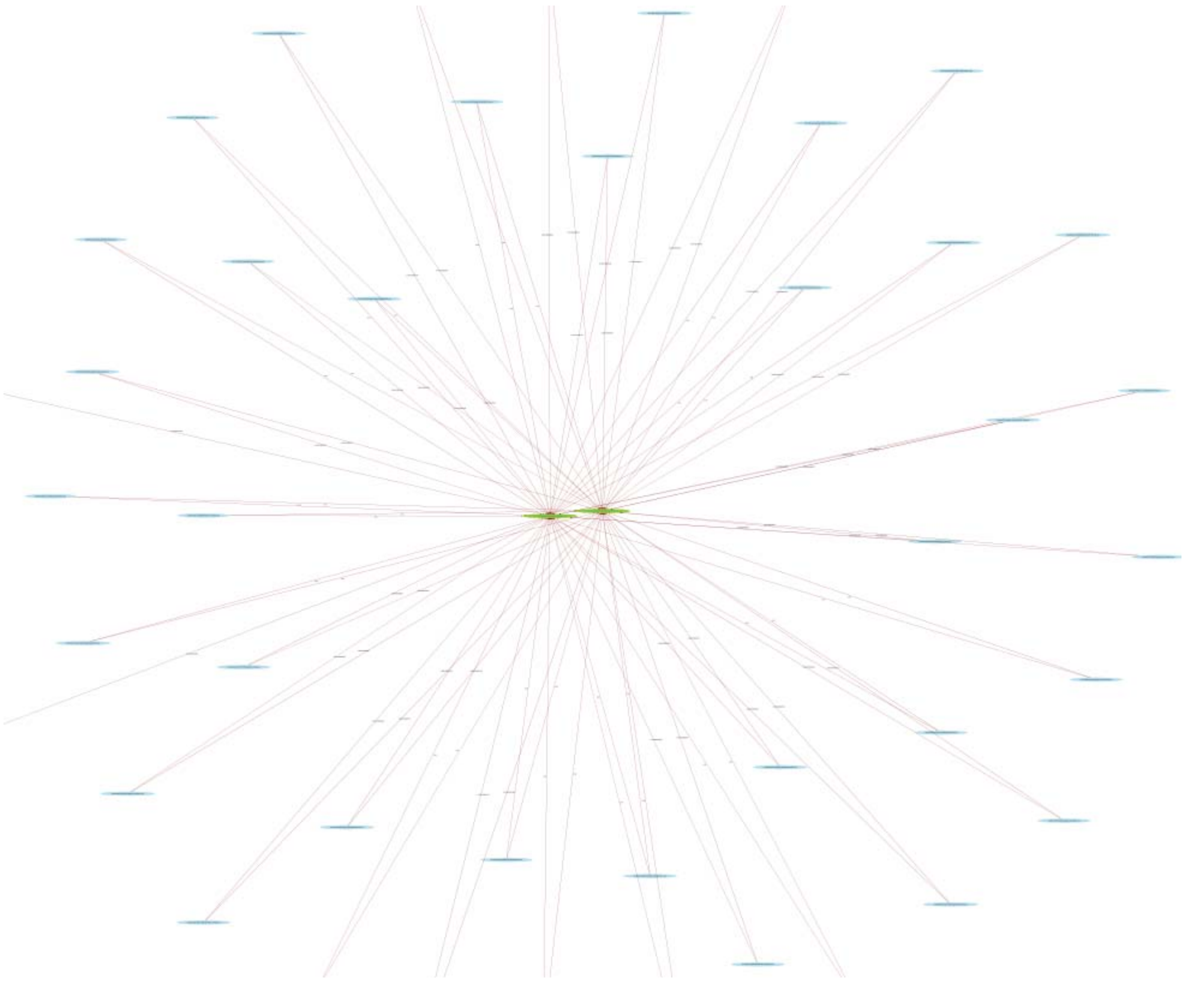


Figure 6.7: A small cluster representing a small class, having 2 exemplars, represented as green nodes and multiple samples of the same class, represented as blue nodes.

from the model for testing?

2. How much does learning new and novel classes increases the overall accuracy of the classifier?

6.4.2.1 Experiment 2: Novelty Detection (Supervised)

A full description of this experiment follows:

1. The model was trained with five classes. This created a model comprising of 9 exemplars (2 exemplars selected from class 1, 3 from class 2, 1 from class 3, 2 from class 4, 1 from class 5).
2. Both exemplars for classes 1 and 4 were removed from the model.

Method	% Accuracy	Outliers
Algorithm 1	70	1300
Algorithm 2	97	0

Table 6.2: Learning with Algorithm 2

3. After classification, the system found that all elements of class 1 and 4 were outliers, and none of classes 1, 2, 3, 5 were outliers.
4. The novelty detection algorithm was applied and three exemplars of class 1 and two of class 4 were added to the model.
5. Thus the classes that were manually removed earlier were automatically detected and added to the model, and no others.

6.4.2.2 Results of Experiment 2

In order to validate Algorithm 2, we used Algorithm 1 to create a model from a small training set (TR) comprising of 50 samples representing five classes, and then deleted the exemplars of two classes from it, as shown in figure 6.8. We then used the first step of Algorithm 1 to test on a dataset that included both known and unknown classes. This resulted in 80% correct classification in terms of accuracy. The unknown samples were appended to the outlier list as shown in Figure 6.9.

Following this, the outlier list was used with Algorithm 2 to create new exemplars for these unknown classes, as illustrated in Figure 6.10.

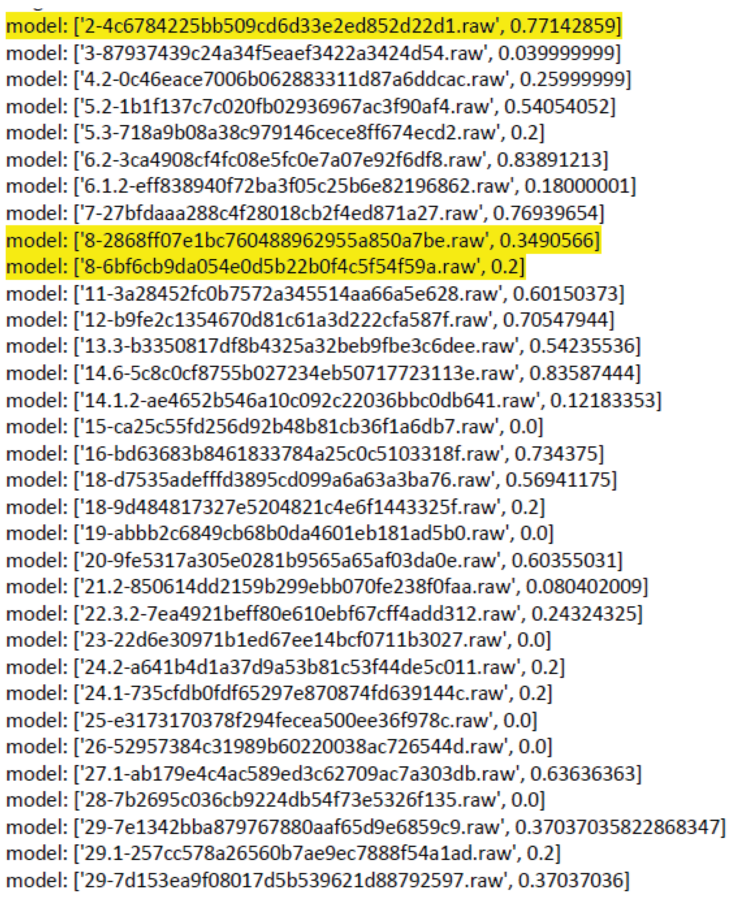
6.4.2.3 Experiment 3: Novelty Detection (Unsupervised)

Finally, we tested the combination of the two algorithms on a larger set of 6600 labelled samples with 50 classes. We started with a small training set (TR), of around 300 samples representing 30 classes. Algorithm 1 was used to learn a model of this dataset, and created 49 exemplars representing 30 classes. The model was then tested on an independent test set of around 6000 samples with all 50 classes represented.

6.4.2.4 Results of Experiment 3

Using the model created by Algorithm 1 (exemplar selection algorithm) to classify the large test set for experiment 3 resulted in an accuracy of approximately 70%. The data points identified as outliers were then learnt about using Algorithm 2 and their exemplars appended to the model. A second test set (of the last 300 data points) was applied, giving classification results of 97% on this second test set, with all 50 classes represented as summarized in table 6.2

These results are very pleasing, and show that this simple method of lifelong learning based on our exemplar learning has distinct promise for detecting and classify new malicious streams for the IDS application.



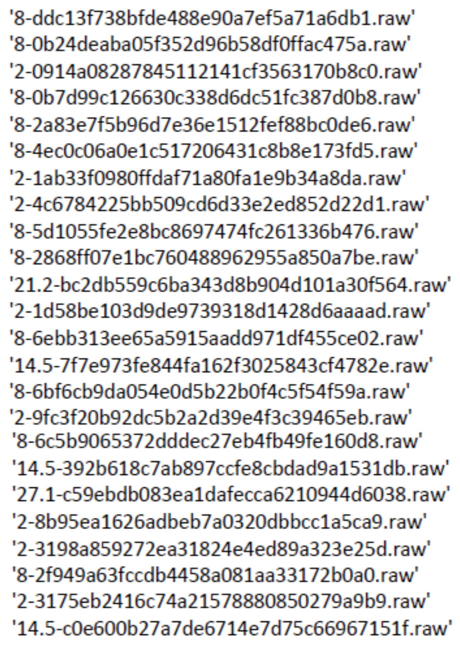
```
model: ['2-4c6784225bb509cd6d33e2ed852d22d1.raw', 0.77142859]
model: ['3-87937439c24a34f5eaef3422a3424d54.raw', 0.039999999]
model: ['4.2-0c46eace7006b062883311d87a6ddcac.raw', 0.25999999]
model: ['5.2-1b1f137c7c020fb02936967ac3f90af4.raw', 0.54054052]
model: ['5.3-718a9b08a38c979146cece8ff674ecd2.raw', 0.2]
model: ['6.2-3ca4908cf4fc08e5fc0e7a07e92f6df8.raw', 0.83891213]
model: ['6.1.2-eff838940f72ba3f05c25b6e82196862.raw', 0.18000001]
model: ['7-27bfdaaa288c4f28018cb2f4ed871a27.raw', 0.76939654]
model: ['8-2868ff07e1bc760488962955a850a7be.raw', 0.3490566]
model: ['8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw', 0.2]
model: ['11-3a28452fc0b7572a345514aa66a5e628.raw', 0.60150373]
model: ['12-b9fe2c1354670d81c61a3d222cfa587f.raw', 0.70547944]
model: ['13.3-b3350817df8b4325a32beb9fbe3c6dee.raw', 0.54235536]
model: ['14.6-5c8c0cf8755b027234eb50717723113e.raw', 0.83587444]
model: ['14.1.2-ae4652b546a10c092c22036bbc0db641.raw', 0.12183353]
model: ['15-ca25c55fd256d92b48b81cb36f1a6db7.raw', 0.0]
model: ['16-bd63683b8461833784a25c0c5103318f.raw', 0.734375]
model: ['18-d7535adefffd3895cd099a6a63a3ba76.raw', 0.56941175]
model: ['18-9d484817327e5204821c4e6f1443325f.raw', 0.2]
model: ['19-abbb2c6849cb68b0da4601eb181ad5b0.raw', 0.0]
model: ['20-9fe5317a305e0281b9565a65af03da0e.raw', 0.60355031]
model: ['21.2-850614dd2159b299ebb070fe238f0faa.raw', 0.080402009]
model: ['22.3.2-7ea4921beff80e610ebf67cff4add312.raw', 0.24324325]
model: ['23-22d6e30971b1ed67ee14bcf0711b3027.raw', 0.0]
model: ['24.2-a641b4d1a37d9a53b81c53f44de5c011.raw', 0.2]
model: ['24.1-735cfdb0fdf65297e870874fd639144c.raw', 0.2]
model: ['25-e3173170378f294fecea500ee36f978c.raw', 0.0]
model: ['26-52957384c31989b60220038ac726544d.raw', 0.0]
model: ['27.1-ab179e4c4ac589ed3c62709ac7a303db.raw', 0.63636363]
model: ['28-7b2695c036cb9224db54f73e5326f135.raw', 0.0]
model: ['29-7e1342bba879767880aaf65d9e6859c9.raw', 0.37037035822868347]
model: ['29.1-257cc578a26560b7ae9ec7888f54a1ad.raw', 0.2]
model: ['29-7d153ea9f08017d5b539621d88792597.raw', 0.37037036]
```

Figure 6.8: Original model, with highlighted samples that will be removed from the model for testing the novelty detection algorithm

6.4.2.5 Discussion

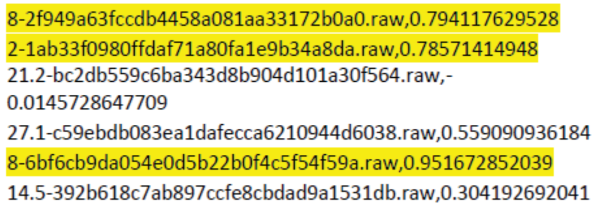
The question that motivated our first validation test was that: can the novelty detection algorithm detect known classes, which were removed from the model for testing?

The first test conducted for validating the algorithm involved creating a model from a small training set, removing exemplars from random classes from this model and then performing classification. This resulted in a partial or no classification for classes



```
'8-ddc13f738bfde488e90a7ef5a71a6db1.raw'
'8-0b24deaba05f352d96b58df0ffac475a.raw'
'2-0914a08287845112141cf3563170b8c0.raw'
'8-0b7d99c126630c338d6dc51fc387d0b8.raw'
'8-2a83e7f5b96d7e36e1512fef88bc0de6.raw'
'8-4ec0c06a0e1c517206431c8b8e173fd5.raw'
'2-1ab33f0980ffdaf71a80fa1e9b34a8da.raw'
'2-4c6784225bb509cd6d33e2ed852d22d1.raw'
'8-5d1055fe2e8bc8697474fc261336b476.raw'
'8-2868ff07e1bc760488962955a850a7be.raw'
'21.2-bc2db559c6ba343d8b904d101a30f564.raw'
'2-1d58be103d9de9739318d1428d6aaaad.raw'
'8-6ebb313ee65a5915aadd971df455ce02.raw'
'14.5-7f7e973fe844fa162f3025843cf4782e.raw'
'8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw'
'2-9fc3f20b92dc5b2a2d39e4f3c39465eb.raw'
'8-6c5b9065372dddec27eb4fb49fe160d8.raw'
'14.5-392b618c7ab897ccfe8cbdad9a1531db.raw'
'27.1-c59ebdb083ea1dafecca6210944d6038.raw'
'2-8b95ea1626adbeb7a0320dbbcc1a5ca9.raw'
'2-3198a859272ea31824e4ed89a323e25d.raw'
'8-2f949a63fccdb4458a081aa33172b0a0.raw'
'2-3175eb2416c74a21578880850279a9b9.raw'
'14.5-c0e600b27a7de6714e7d75c66967151f.raw'
```

Figure 6.9: List of outliers found by the novelty detection algorithm



```
8-2f949a63fccdb4458a081aa33172b0a0.raw,0.794117629528
2-1ab33f0980ffdaf71a80fa1e9b34a8da.raw,0.78571414948
21.2-bc2db559c6ba343d8b904d101a30f564.raw,-
0.0145728647709
27.1-c59ebdb083ea1dafecca6210944d6038.raw,0.559090936184
8-6bf6cb9da054e0d5b22b0f4c5f54f59a.raw,0.951672852039
14.5-392b618c7ab897ccfe8cbdad9a1531db.raw,0.304192692041
```

Figure 6.10: New Exemplar ϵ_{novel} and Threshold τ_{novel} pairs proposed by the algorithm. The highlighted samples/exemplars are the samples from the same class that we initially removed from the original model

that were removed from the model and a large list of outliers. These outliers were then run through the novelty detection algorithm to yield results. The results were quite satisfactory as the algorithm correctly identified the missing classes and proposed respective exemplars for their classification.

Our second validation test was motivated by the question: How much does learning new and novel classes increases the overall accuracy of the classifier?

For this test, we created a model from a training set of approx 300 samples extracted from a large dataset of 6600 samples. The model was used to classify the test set, comprising of 6600 samples. After classification, the classifier gave an overall accuracy of 90%. A list outliers was identified and the novelty detection algorithm was then applied to them to identify novel classes and exemplars. These new exemplars were then appended to the model for re-classification. The new classification resulted in a an overall accuracy of 98%.

6.5 Comparative work

Using a small labelled test set comprising of 1200 samples and a training set of 300 samples, we performed a comparative analysis of the exemplar learning technique with both of our prior techniques i.e. (a) Classification using a fixed threshold (T-NN Fixed) and (b) Classification at the optimal threshold obtained by adaptive thresholding (T-NN Adaptive). Both of these techniques provide a good approximation, but do not give a precise measurement in terms of: (a) the optimum number of exemplars that are required per class and (b) the optimum similarity threshold for those exemplars in that class. Next, these results will be compared with well-known machine learning algorithms like nearest neighbour and k-nearest neighbour, which do not require threshold constraints, and finally concluded with a comparison using our proposed exemplar-learning and novelty detection algorithm.

6.5.1 Experiment 4: Instance Selection

For comparison we performed a two-stage experiment to answer the following questions:

6.5.1.1 How best to select the exemplars or instances required per class for classification?

Previously, for randomly selecting exemplars per class from the training set TR, we sampled on a smaller scale by randomly selecting 1 to 9 exemplars from the training set (TR) in increments of 1. For this experiment the exemplars were selected in step-wise increments of 10 exemplars ($\epsilon = \epsilon + 10$), picked randomly from the training set starting from $\epsilon = 10$ till $\epsilon = 90$ exemplars. In each iteration, a set of exemplars were produced that were then subjected to the second stage. The second stage answers the following question.

6.5.2 Experiment 5: Threshold Selection

6.5.2.1 How to select the optimal threshold required per instance per class?

To answer this question we used an adaptive threshold method that requires small step-wise threshold increments of $\tau = \tau + 0.25$ starting from 0.0 till 1.0 creating a confusion matrix for each class individually and determining the optimum threshold τ for the whole test set.

6.5.3 Comparative Results

For comparison of the classification techniques, we divided the methodologies into three main categories:

1. Methodologies that involve threshold for classification or threshold-based e.g. fixed and adaptive thresholds.
2. Methodologies that involve just instances for classification or instance-based learning e.g. nearest neighbour and k-nearest neighbour
3. A hybrid methodology that uses both instance-based learning guided by the threshold for classification e.g. threshold-based exemplar-learning techniques

Accuracy results for (1) fixed threshold (T-NN Fixed) and (2) optimized thresholds (T-NN Adaptive) (3) nearest neighbour and (4) k-nearest neighbour at k = 3 and 5 were plotted using an XY Scatter plot for each exemplar set. Additionally, results from our proposed exemplar-learning and novelty detection algorithm were added for comparison.

The results from this experiment are shown in Figure 6.11, it can be seen that as the number of exemplars increases, it increases the accuracy since the chances of getting a better classification increases with it, due to better coverage. This however does not guarantee the best or optimal exemplar selection for each class, as the exemplars are picked randomly from the training set. The performance of the (a) optimized threshold technique is better than (b) the fixed threshold, as enumerating through every possible result to yield the best one, on average, gives approximately 8% better results. Results from nearest neighbour and the k-nearest neighbour algorithm have been added to the results for comparison. Here it can be seen that the nearest neighbour algorithm performs very well as the number of exemplars increases and is found to be better than (a) and (b). For the k-nearest neighbour algorithm, we are not certain about the value of k, so we used both k = 3 and 5 respectively. It can be observed here and also in table 6.3, that as the size of k increases, the accuracy decreases as it is believed to add more false positives to the results, which eventually have a greater vote and thus detrimentally impacts on the accuracy of the classifier. Finally, results of exemplar learning (Algorithm 1) have been added to the graph. It can be seen in Figure 6.11 that using Algorithm 1 results in 30 exemplars with a percentile accuracy

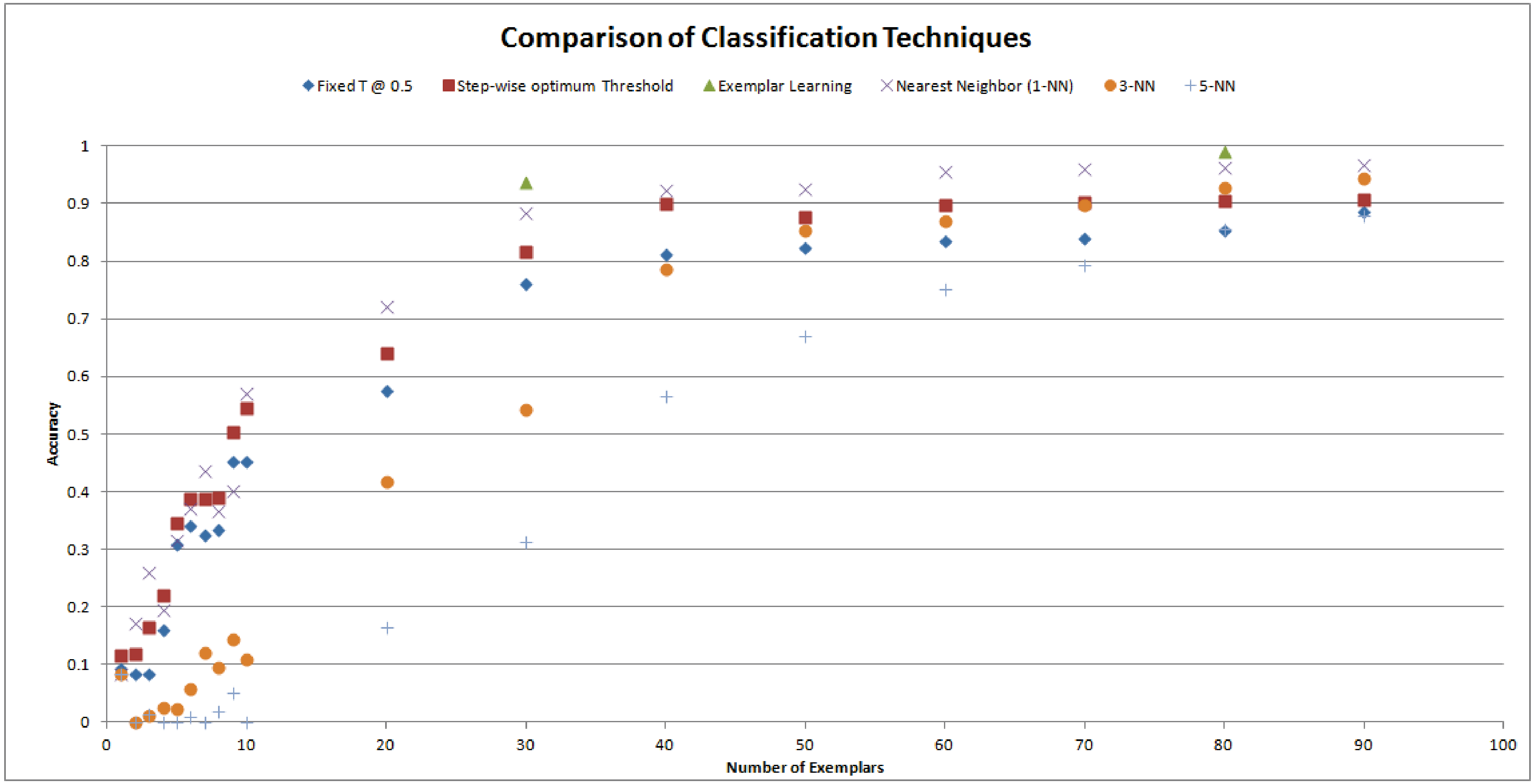


Figure 6.11: A comparison of classification methodologies showing accuracy results for: (1) Fixed threshold (2) Optimum threshold using step-wise thresholding (3) Nearest Neighbour (4) k-Nearest Neighbour and (5) for exemplar learning. Results are calculated at varying number of exemplars for techniques 1-4. It can be seen here that exemplar learning results using Algorithm 1 (first green triangle) and then Algorithm 2 (second green triangle) outperforms other techniques by yielding better accuracy with least number of exemplars.

Exemplars	Accuracy @ Fixed Threshold	Accuracy using adaptive threshold	Accuracy using Nearest Neighbour	Accuracy using k-NN, k=3	Accuracy at k=5
10	0.452157	0.544964	0.571024	0.109010	0
20	0.576236	0.640812	0.721024	0.417137	0.1642226
30	0.759805	0.817226	0.884628	0.5422261	0.3131625
40	0.811627	0.900676	0.923233	0.78524	0.565282
50	0.822666	0.877459	0.925088	0.853533	0.66934
60	0.835433	0.898047	0.955778	0.869257	0.752385
70	0.83998	0.903421	0.959452	0.898144	0.792667
80	0.854127	0.904411	0.963250	0.926855	0.856713
90	0.885432	0.907568	0.967844	0.944169	0.878091

Table 6.3: Comparison of algorithms using incremental numbers of exemplars

of approximately 90%, with some outliers. Next we apply Algorithm 2 on the outliers which results in approximately 80 exemplars with an overall 98% accuracy result.

6.5.4 Comparison with other instance selection algorithm

Instance selection algorithms can be classified into three main groups namely, edition algorithms, condensation algorithms and hybrid algorithms ((Aha et al., 1991) and (Garcia et al., 2012)).

Edition algorithms edit out noise instances as well as close border class. e.g ENN[2],

Condensation algorithms aim to condense the selection set from the training set, while retaining points close to the class boundary. e.g. CNN[3], RNN[4], SNN[5]

Hybrid algorithms aim to increase classification accuracy by using a combination of approaches from edition and condensation algorithms. e.g. instance-based learning family of algorithms IB3. Our proposed instance selection algorithm (Algorithm 1) is a hybrid algorithm as it performs both edition and condensation at a per class level to select the the most appropriate instances for creating the model.

6.5.4.1 Difference between ENN and our proposed algorithm

ENN is an edition algorithm which uses a decremental search. ENN starts by considering the entire selection set as the training set i.e. S = TR and then removes all such points/selections which have a negative class neighbour majority discovered by k-nn.

[2]Edition-based Nearest Neighbour
[3]Condensed Nearest Neighbour
[4]Reduced Nearest Neighbour
[5]Selective Nearest Neighbour

Our proposed exemplar selection algorithm (Algorithm 1) is a hybrid instance selection algorithms, which starts with an empty selection set i.e. S = [ ]. After that it creates an $n \times n$ similarity matrix of all points in the training set (TR), then for each class it finds the point with the shortest distance to all the other points in that same class/category. This point is chosen to be the initial instance or the exemplar for that class, and is added to the list of selections. Next an optimal threshold is calculated in a way to avoid any possible false positives i.e. points/samples from other classes. For any threshold if there are FP observations, the threshold value is reduced. This process continues till a threshold with no FP values is selected. This threshold is also added to the selection list in correspondence to its exemplar or instance. The process is iterated for the remainder points/prototypes in the class, untill we have enough instances/exemplars and thresholds to classify the training set with maximum accuracy and minimum exemplars. Compared to ENN this method can guarantee a comparatively small number of exemplars/instances that are required for classification, as shown in table 6.4.

Differences with ENN can be highlighted as follows:

1. Starts with S = [ ]
2. Does not use k-NN
3. Uses a similarity matrix
4. Computation and instance selection is done on a per class basis
5. Selects fewer exemplars

6.5.4.2 Experiment 6: Comparison with other Instance Selection Algorithms (ENN)

We conducted an experiment to empirically compare the performance of ENN with our proposed model selection algorithm:

1. A training set (TR) was created comprising of 334 samples. The samples were selected randomly from the dataset using the criteria of 10 or less random samples or instances per class.
2. ENN was implemented on this training set TR to select instances/exemplars.
3. Next, our proposed instance selection algorithm (Algorithm 1) was implemented on the training set TR to select instances/exemplars.
4. Compared the results.

Algorithm	Exemplars/Instances Selected	% reduction in TR
ENN	320	4.2
Algorithm 1	36	88.75

Table 6.4: Comparison with ENN

6.5.4.3 Results of Experiment 6

1. Out of a total of 334 samples, ENN was able to reduce the size of the TR by 4.2%, resulting in a total of 320 instances or a selection set comprising of approx 95% of the TR.

2. The exemplar selection algorithm selected 36 instances/exemplars out of the 334, thus reducing the TR by approximately 89% with a selection set comprising of only about 11% of the TR. With this model, the classifier was able to achieve an overall accuracy of approx 90% on a large test set of 6000 samples. A small TR would result in a more compact model. This would improve the classifiers performance significantly as fewer similarity comparison operations would be required. However, the case would be different if the model comprised of many exemplars, as each test set sample/datapoint would be required to be tested for similarity with each exemplar sample/datapoint.

6.6 Conclusion

It is concluded that exemplar-selection approaches can be applied effectively to select the best exemplar and threshold from any given class. The exemplars can be used as a reference signature and added to the model for classification. It was observed that the classification accuracy increased manifold by applying the proposed exemplar-learning algorithm on supervised data. The accuracy was increased even further with the application of the algorithm that deals with exemplar-learning for novel data. It was demonstrated that novel classes can be identified and exemplars from these novel classes can be selected intelligently. It was demonstrated that an exemplar-learning algorithm with novelty detection has comparatively given the best results in terms of accuracy of the classification, followed by nearest neighbour and optimized or adaptive thresholding.

Chapter 7

Conclusions and Future Work

"The only thing necessary for the triumph of evil is for good men to do nothing."
– *Edmund Burke*

This chapter provides an overview of the research done in this thesis, presents a conclusion to the research questions addressed and highlights the novel contributions to intrusion detection. This chapter concludes with a discussion of directions for future work.

7.1 Summary of the Research and Significant findings

The objective of this thesis is the detection and classification of known malicious network streams, their variants, and novel streams. Samples of malicious streams are collected using Honeynets and labelled. Techniques to determine similarity between malicious streams are investigated. Clustering and classification algorithms are used for clustering and classification of similar malicious streams. Supervised learning approaches are used for classification of known malicious streams and their variants, while unsupervised learning approaches are used for the classification of new or novel malicious or suspicious streams.

The thesis begins with an introduction and background to the information security problem. This is followed by the motivation for this thesis, which highlights the current threat landscape, reports some significant cyber intrusions and incidents, identifies problems with existing information security tools and lists research areas to investigate. Next, the challenges in network-based intrusion detection schemes are identified and the need for intrusion detection and classification of malicious network streams is described. After that, the scope of the research is presented and research problems are categorized and highlighted. The research goals are identified followed by an account of the contributions and limitations of this thesis. The chapter concludes with an outline of this thesis.

The literature review reviews the state of the art in intrusion detection methodologies and technologies, their history, types and taxonomy and their strengths and weaknesses. The detection schemes are classified and each class is discussed with related work. This review revealed a shift from simple statistical anomaly-based intrusion detection systems, towards more complex and intelligent systems that incorporate data-mining and machine learning methodologies. It was demonstrated that existing methods fail to satisfy the criteria to detect attack variants and novel attacks mentioned in this thesis. It was shown that many intrusion detection techniques focused mainly on features extracted from packet headers alone, while the actual exploit or attack data is sent through the payload. Techniques such as signature-based detection only look for certain patterns or strings in certain locations of the payload. Such systems fail to detect small variations of the same attack, resulting in false-negatives (FN). It was shown that information-theoretic methods can be applied to intrusion detection to detect similar traffic and is more robust and less error prone to other methods. Machine learning methods can be used to add lifelong learning to the system. After a review of IDS, computer attacks are studied to label and categorize attack samples in datasets. Data collection techniques to test and train IDSes were discussed. A critical analysis of existing datasets was presented and Honeypots and Honeynets were introduced as real data collection tools and a means to study attacks and attackers.

After the literature review, a detailed design and architecture of our proposed IDS is presented, followed by a discussion on the datasets that were collected and used for evaluating the IDS. A discussion on Honeypot and Honeynet deployment strategies, design, architecture and implementation challenges are also presented. Improvements to existing Generation III design for virtual Honeynet deployment were suggested. Results are presented which demonstrate advantages of using Honeynets as they provide extra insight into intrusions, which other tools lack. This is mainly due to the assumption that since the Honeypot has no production value and is not advertised to the outside world, any interaction in the form of traffic directed towards it is assumed to suspicious and malicious in nature, thus they actively collect filtered traffic for the researcher. Additionally, the Honeynet architecture allows a researcher to control both the intermediate network and the endpoint (Honeypot).

Next, a foundation for the experimental approach of this thesis is presented, in which, similarity measurement techniques are identified, followed by a discussion on how edit distance and information theoretic similarity metrics can be used to determine similarity between malicious network streams and their variants. The problems and challenges are determined. Various similarity metrics were reviewed and some were chosen from the literature, that have been used to determine the similarity between various packet or stream profiles. A methodology is developed in response to these challenges, followed by a series of experiments to evaluate these metrics given different labelled real and synthesized payload samples. The results for these experiments are presented. Finally, validation and comparison strategies are investigated with a conclusion to suggest the best metric for comparison. This study revealed that a hybrid metric using both NCD and Spamsum proved to be the best metric to accurately measure the similarity of malicious streams. This work was also compared with the study done by (Eiland and Liebrock, 2006) who used inverse compression ratio to detect scan traffic. It was concluded that our proposed novel hybrid approach is more accurate.

The research questions addressed include:

1. Is it possible to measure similarity between samples of malicious network streams?
2. What metrics are available?
3. How do these metrics compare?
4. What metrics provide the most appropriate methods for making similarity comparisons?
5. Is it possible to measure similarity between variants of malicious network streams with statistical confidence?

From the results, it be determined that instead of matching strings in the payload, packet payload information on the whole can be used as a reference signature to classify malicious streams using network data. Therefore, it is possible to perform similarity measurements on packet payload data with confidence. The system proved to be robust and allows for significant changes in the signatures without misclassifying the stream. For this type of data, both NCD and Spamsum metrics are the most appropriate similarity metrics. Although both metrics use different methodologies for similarity calculation and have some advantages over the other, a high correlation was observed in their results. Based on this criteria it was decided to use them in a hybrid approach for similarity measurement.

Once a means to measure similarity had been established, machine learning techniques were investigated to cluster and classify malicious network streams. Some machine learning approaches for intrusion detection are discussed. Clustering and Classification algorithms are studied and one is proposed for our domain. Experiments are designed to find the best clustering algorithm and cut-off threshold for our domain. The results are analysed with a discussion on ROC-AUC based measures and Accuracy. Results obtained by comparing all the clustering algorithms used for the evaluation are discussed and the best ones are highlighted. Here the following research questions were addressed:

1. How should similar items be grouped together?
2. How should optimal threshold to partition data into groups be determined?
3. How to find the best clustering algorithm(s) for our domain?
4. How to evaluate the results for classifier correctness?
5. How to identify the most suitable similarity metric for our domain using the above evaluations?

The use of machine learning techniques like clustering, on results obtained from information theoretic similarity metrics, can be used for classification of malicious streams using network payload data. Threshold plays a significant role in clustering and classification and determining an optimal threshold is a challenging task. We found that for this type of data, NCD and Spamsum metrics are the most appropriate similarity metrics and ROC-AUC is a better measure than Accuracy.

Next, the most suitable clustering algorithms for our domain were identified from the literature. After a theoretical comparison of these algorithms, a custom algorithm was proposed. Finally all these clustering algorithms were compared with each other using a standard dataset and after extensive testing, it was revealed that threshold based nearest neighbour or T-NN serves as the best algorithm for clustering and classification. Later, some limitations of T-NN were also identified.

At this stage we investigate methodologies to automatically select the best exemplar from each class to create a model. For this purpose, exemplar-selection algorithms are studied to select the most important exemplars and threshold from the training set. These are used to create models to guide the classification. Machine learning techniques are used to automatically choose the best exemplars and threshold from each class, along with methods to make our system evolve by learning exemplars from novel classes. Life-long learning strategies from an implementation perspective are investigated, highlighting algorithms for supervised and unsupervised learning for intrusion detection. Customized exemplar learning algorithms are proposed for our domain for both supervised and unsupervised learning approaches. These algorithms are validated experimentally. These proposed algorithms are then compared with existing clustering and classification algorithms discussed in previous chapter for evaluation. Additionally, approaches involving automated learning of exemplar signature and threshold for optimum classification of known and novel classes are studied and compared. Here the following research questions were answered:

Given the data and the similarity metric, there are four problems for the exemplar learner.

1. How to identify and select suitable exemplars or instances from the training set (TR)(Instance selection problem)?
2. How to learn an appropriate similarity threshold within which the data points belong to the same class as the exemplar class, for better classification?
3. How to identify unknown or novel samples that did not get classified by prior knowledge extracted from the training set (TR)?
4. How to select appropriate exemplars from these unknown or novel samples for classification?
5. How to Identify groups and sub-groups that may exist in these novel samples and perform classification?

It is concluded that exemplar-selection approaches can be applied effectively to select the best exemplar and threshold from any given class. The exemplars can be used as a reference signature and added to the model for classification. It was observed that the classification accuracy increased substantially by applying the proposed exemplar-learning algorithm on supervised data. The accuracy was increased even further with the application of the algorithm that deals with exemplar-learning for novel data. It was demonstrated that novel classes can be identified and exemplars from these novel classes can be selected intelligently. It was demonstrated that the exemplar-learning algorithm with novelty detection has comparatively given the best results in terms of accuracy of the classification, followed by nearest neighbour and optimized or adaptive T-NN.

7.2 Implementation summary

The current system is implemented as an offline system, which first captures the malicious streams using Honeypots and then sends the daily captured streams for analysis and detection. These streams (collected over a period of 24 hours) are then analysed using a model of existing exemplars for the detection of malicious streams, variants of malicious streams and novel streams. New exemplars selected from novel streams are labelled using random alpha-numeric labels and are then added to the model. The process continues and new knowledge in the form of novel exemplars are added to the model to enhance the classifier's performance. It has been demonstrated that the proposed technique has significantly improved the detection results with a smaller number of exemplars than currently used techniques. Additionally, with life-long learning capabilities, the system continually learns novel exemplars in an automated and unsupervised manner and adds them to the existing knowledge-base to enhance the detection, without human intervention.

7.3 Contribution of this thesis

1. Developed a means to contain and capture attacker activity in a controlled environment.

2. Developed a method to reliably measure the similarity between malicious and benign network streams with statistical confidence.

3. Developed the capability to detect variants of known malicious streams with confidence and classify them.

4. Developed the capability to detect new and novel streams.

5. Developed the capability to classify these novel streams into groups and subgroups.

7.4 Future Research and Open Questions

For future research we would like to extend our system. Various possibilities are listed below:

- The system proposed in this thesis could be integrated into Dionaea Honeypot[1] as a detection and analysis sub-system. Detection and analysis of incoming streams would begin as soon as they arrive.

[1]`http://dionaea.carnivore.it/`

- The classification of novel streams will result in novel exemplars. These exemplars could be automatically labelled using input from free malware detection and analysis services such as VirusTotal[2], automated sandbox environments such as Anubis[3], Joe Sandbox[4], Norman Sandbox[5] and Cuckoo[6]. The exemplars and the analysis reports could also be sent to a human analyst to create signatures for Snort or other signature-based IDS.

- Methodologies that may involve similarity metrics for similarity measurements, followed by analysis of the data to determine the best approach for clustering, classification, life-long learning and novelty detection can be investigated further, to detect and classify:

  - compressed malicious network streams. This work would involve the creation of a labelled dataset of compressed malicious streams for evaluation.
  - binary malware. A labelled dataset of binary malware could be created for evaluation. To validate these labels, free malware detection and analysis services like VirusTotal and automated sandbox environments as discussed in the first point, can be used and integrated with the system.
  - encrypted malicious network streams, encrypted with substitution ciphers. A labelled dataset of encrypted malicious streams could be created for evaluation. The basis for this work would be the hypothesis that the same encryption algorithm should generate the same output, as long as the encryption key remains constant. Several permutations of the same malicious sample can be created.
  - system events from a host-based intrusion detection perspective. This work would involve the creation of a labelled dataset of system-wide malicious event profiles for evaluation. Honeypots that can track or log changes to system, services, file system, registry, memory and network sockets could be of great value for this purpose.

- Metasploit is a widely used exploit framework used by hackers and penetration testers. It can be used to create datasets of known exploits and shellcode. Training the system on such datasets could possibly give more detection coverage to the system. Other frameworks such as MACE (Sommers et al., 2004) and FLAME (Brauckhoff et al., 2008) may also be used.

[2]`http://www.virustotal.com`
[3]`http://www.anubis.iseclab.org`
[4]`www.joesecurity.org`
[5]`http://www.norman.com/security_center/security_tools/submit_file/en`
[6]`http://www.cuckoosandbox.org/`

- Explore alternative similarity metrics for use in the similarity measurements.
- Investigate means to make the system real-time. This could involve the use of special hardware circuits, parallel processing and optimization using GPU's and FPGA's.
- The similarity measurement approach used in this thesis could have applications in plagiarism detection and also be used to investigate evolutionary data for phylogeny.

Bibliography

Abbasi, F. and Harris, R. (**2010**). "Intrusion detection in Honeynets by compression and hashing." In "Australasian Telecommunication Networks and Applications Conference (ATNAC), 2010," pages 96 –101.

Abbott, R. P., Chin, J. S., Donnelley, J. E., Konigsford, W. L., Tokubo, S., and Webb, D. A. (**1976**). "Security analysis and enhancements of computer operating systems." Technical report, DTIC Document.

Aha, D., Kibler, D., and Albert, M. (**1991**). "Instance-based learning algorithms." *Machine Learning*, 6: 37–66. doi:10.1007/BF00153759.

Allen, J., Christie, A., Fithen, W., McHugh, J., and Pickel, J. (**2000**). "State of the practice of intrusion detection technologies." Technical report, DTIC Document.

Amer, S. and Hamilton Jr, J. (**2010**). "Intrusion Detection Systems (IDS) Taxonomy-A Short Review." *Defense Cyber Security*, 13(2).

Anderson, D., Lunt, T., Javitz, H., Tamaru, A., and Valdes, A. (**1995**). *Detecting unusual program behavior using the statistical component of the Next-generation Intrusion Detection Expert System (NIDES).* SRI International, Computer Science Laboratory.

Anderson, J. P. (**1980**). "Computer security threat monitoring and surveillance." Technical report, Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

Argus Project, . (**2008, February**). "Argus Project." Retrieved from `http://qosient.com/argus/`. Accessed: 2010-09-30.

Ariu, D. and Giacinto, G. (**2010**). "HMMPayl: an application of HMM to the analysis of the HTTP Payload." In "workshop on applications of pattern analysis, Cumberland Lodge, 2010. Proceedings Cumberland Lodge, WAPA," pages 81–87.

Aslam, T. (**1995**). *A taxonomy of security faults in the unix operating system.* Ph.D. thesis, Purdue University.

Axelsson, S. (**2000**). "Intrusion Detection Systems: A Survey and Taxonomy." Technical report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Balas, E. and Viecco, C. (**2005**). "Towards a third generation data capture architecture for honeynets." In "Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC," pages 21–28. IEEE.

Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., and Zamboni, D. (**1998**). "An architecture for intrusion detection using autonomous agents." In "Computer Security Applications Conference, 1998. Proceedings. 14th Annual," pages 13–24. IEEE.

Barbara, D., Wu, N., and Jajodia, S. (**2001**). "Detecting novel network intrusions using bayes estimators." In "First SIAM Conference on Data Mining," Citeseer.

Bisbey, R. and Hollingworth, D. (**1978**). "Protection Analysis." Technical report, USCISI/SR-78-13, May.

Bloedorn, E., Christiansen, A. D., Hill, W., Skorupka, C., Talbot, L. M., and Tivel, J. (**2001**). "Data mining for network intrusion detection: How to get started." Technical report, MITRE.

Bolzoni, D., Etalle, S., and Hartel, P. (**April**). "POSEIDON: a 2-tier anomaly-based network intrusion detection system." In "Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on," pages 10–156.

Brauckhoff, D., Wagner, A., and May, M. (**2008**). "FLAME: a flow-level anomaly modeling engine." In "Proceedings of the conference on Cyber security experimentation and test," pages 1–6. USENIX Association.

Bridges, S. M. and Vaughn, R. B. (**2000**). "Intrusion detection via fuzzy data mining." In "12th Annual Canadian Information Technology Security Symposium," pages 109–122.

Cavallaro, L., Lanzi, A., Mayer, L., and Monga, M. (**2008**). "LISABETH: automated content-based signature generator for zero-day polymorphic worms." In "inproceedings of the fourth international workshop on Software engineering for secure systems," pages 41–48. ACM.

CERIAS (**2011**). "CERIAS." `http://www.cerias.purdue.edu/about/history/coast_resources/intrusion_detection/`. Accessed: 30/09/2010.

Chandola, V., Banerjee, A., and Kumar, V. (**2009**). "Anomaly detection: A survey." *ACM Computing Surveys (CSUR)*, 41(3): 1–58.

Chebrolu, S., Abraham, A., and Thomas, J. P. (**2005**). "Feature deduction and ensemble design of intrusion detection systems." *Computers & Security*, 24(4): 295 – 307. doi:10.1016/j.cose.2004.09.008.

Chen, W.-H., Hsu, S.-H., and Shen, H.-P. (**2005**). "Application of SVM and ANN for intrusion detection." *Comput. Oper. Res.*, 32(10): 2617–2634. doi:10.1016/j.cor.2004.03.019.

Cilibrasi, R. (**2003**). "The complearn toolkit, 2003." *IEEE Transactions on Information Theory*, 3(2).

Cilibrasi, R. and Vitányi, P. (**2005**). "Clustering by compression." *Information Theory, IEEE Transactions on*, 51(4): 1523–1545.

Cohen, W., Ravikumar, P., and Fienberg, S. (**2003**). "A comparison of string distance metrics for name-matching tasks." In "Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)," pages 73–78. Citeseer.

Crosbie, M., Dole, B., Ellis, T., Krsul, I., and Spafford, E. H. (**1996**). "Idiot-users guide." Technical report.

Debar, H., Dacier, M., and Wespi, A. (**1999**). "Towards a taxonomy of intrusion-detection systems." *Computer Networks*, 31(8): 805–822.

DeLooze, L. (**2004**). "Classification of computer attacks using a self-organizing map." In "Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC," pages 365 – 369.

Denning, D. (**1987**). "An Intrusion-Detection Model." *Software Engineering, IEEE Transactions on*, SE-13(2): 222–232.

Dionaea (2012, February). "Dionaea catches bugs. Onilne." `http://dionaea.carnivore.it/`.

Eiland, E. and Liebrock, L. (**2006**). "An application of information theory to intrusion detection." In "Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on," pages 126–134.

Ellson, J., Gansner, E., Koutsofios, L., North, S., and Woodhull, G. (**2002**). "Graphviz open source graph drawing tools." In "Graph Drawing," pages 594–597. Springer.

Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.-N., Kumar, V., Srivastava, J., and Dokas, P. (**2004**). "Minds: minnesota intrusion detection system." *Next Generation Data Mining*, pages 199–218.

Evans, S. and Barnett, B. (**2002**). "Network security through conservation of complexity." In "MILCOM," volume 2, pages 1133–1138. IEEE.

Evans, S., Barnett, B., Bush, S. F., and Saulnier, G. J. (**2004**). "Minimum description length principles for detection and classification of FTP exploits." In "Military Communications Conference," volume 1, pages 473–479. IEEE.

Feinstein, L., Schnackenberg, D., Balupari, R., and Kindred, D. (**2003**). "Statistical approaches to DDoS attack detection and response." In "DARPA Information Survivability Conference and Exposition, 2003. Proceedings," volume 1, pages 303 – 314.

Flake, H. (**2004**). "Structural comparison of executable objects." In "Proc. of the International GI Workshop on Detection of Intrusions and Malware & Vulnerability Assessment, in Lecture Notes in Informatics," pages 161–174.

Fogarty, J., Baker, R. S., and Hudson, S. E. (**2005**). "Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction." In "Proceedings of Graphics Interface 2005," GI '05, pages 129–136. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. Retrieved from `http://dl.acm.org/citation.cfm?id=1089508.1089530`.

Gailly, J. and Adler, M. (**1995**). "Zlib home page." Accessed: 2010-09-30.

Garcia, S., Derrac, J., Cano, J., and Herrera, F. (**2012**). "Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3): 417 –435.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (**2009**). "Anomaly-based network intrusion detection: Techniques, systems and challenges." *Computers & Security*, 28(1): 18–28.

Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. (**2007**). "Bothunter: Detecting malware infection through ids-driven dialog correlation." In "Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium," page 12. USENIX Association.

Hammersley, E. (**2006**). *Professional VMware Server.* John Wiley & Sons.

Hansman, S. and Hunt, R. (**2005**). "A taxonomy of network and computer attacks." *Computers & Security*, 24(1): 31 – 43. doi:10.1016/j.cose.2004.06.011.

Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., and Wolber, D. (**1990**). "A network security monitor." In "Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on," pages 296–304. IEEE.

Hoglund, G. and McGraw, G. (**2004**). *Exploiting Software: How to break code.* Pearson Education India.

Howard, J. D. (**1997**). "An analysis of security incidents on the Internet 1989-1995." Technical report, DTIC Document.

Ilgun, K. (**1993**). "USTAT: A real-time intrusion detection system for UNIX." In "Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on," pages 16–28. IEEE.

Innella, P. (**2000**). "A brief history of network security and the need for adherence to the software process model." *Tetrad Digital Integrity.*

Javitz, H. S. and Valdes, A. (**1991**). "The SRI IDES statistical anomaly detector." In "Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on," pages 316–326. IEEE.

Jones, A., Sielken, R., et al. (**2000**). "Computer system intrusion detection: A survey." *Computer science technical report.*

Julisch, K. (**2002**). "Clustering intrusion detection alarms to support root cause analysis." *ACM SIGKDD*, 2(3): 111–138.

Kendall, K. (**1999**). *A database of computer attacks for the evaluation of intrusion detection systems.* Ph.D. thesis, Massachusetts Institute of Technology.

Kim, H. and Karp, B. (**2004**). "Autograph: Toward automated, distributed worm signature detection." In "inproceedings of the 13th conference on USENIX Security Symposium-Volume 13," pages 19–19. USENIX Association.

Kornblum, J. (**2006**). "Identifying almost identical files using context triggered piecewise hashing." *Digital investigation*, 3: 91–97.

Kreibich, C. and Crowcroft, J. (**2004**). "Honeycomb: creating intrusion detection signatures using honeypots." *ACM SIGCOMM Computer Communication Review*, 34(1): 51–56.

Krsul, I. V. (**1998**). *Software vulnerability analysis*. Ph.D. thesis, Purdue University.

Kruegel, C. and Toth, T. (**2000**). "A survey on intrusion detection systems." In "TU Vienna, Austria," Citeseer.

Kruegel, C. and Toth, T. (**2003**). "Using Decision Trees to Improve Signature-Based Intrusion Detection." In G. Vigna, C. Kruegel, and E. Jonsson, editors, "Recent Advances in Intrusion Detection," volume 2820 of *Lecture Notes in Computer Science*, pages 173–191. Springer Berlin Heidelberg. doi:10.1007/978-3-540-45248-5.

Kulkarni, A. and Bush, S. (**2001**a). "Active network management and kolmogorov complexity." In "IEEE OpenArch," pages 27–28.

Kulkarni, A. and Bush, S. (**2001**b). "Active network management and kolmogorov complexity." In "inproceedings of IEEE OpenArch," pages 27–28.

Kulkarni, A. and Bush, S. (**2006**). "Detecting distributed denial-of-service attacks using kolmogorov complexity metrics." *Journal of Network and Systems Management*, 14(1): 69–80.

Kumar, S. (**1995**). *Classification and detection of computer intrusions*. Ph.D. thesis, Purdue University.

Kvarnström, H. (**1999**). "A survey of commercial tools for intrusion detection." Technical report, Technical Report TR99-8, Chalmers University of Technology.

Lakhina, A., Crovella, M., and Diot, C. (**2005**). "Mining anomalies using traffic feature distributions." In "Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications," SIGCOMM '05, pages 217–228. ACM, New York, NY, USA. doi:10.1145/1080091.1080118.

Landwehr, C. E., Bull, A. R., McDermott, J. P., and Choi, W. S. (**1994**). "A taxonomy of computer program security flaws." *ACM Computing Surveys (CSUR)*, 26(3): 211–254.

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., and Srivastava, J. (**2003**). "A comparative study of anomaly detection schemes in network intrusion detection." In

"Proceedings of the third SIAM international conference on data mining," volume 3, pages 25–36. Society for Industrial & Applied.

Lazarevic, A., Kumar, V., and Srivastava, J. (**2005**). "Intrusion detection: A survey." *Managing Cyber Threats*, pages 19–78.

Lee, W., Stolfo, S., and Mok, K. (**1999**). "A data mining framework for building intrusion detection models." In "Proceedings of the 1999 IEEE Symposium on Security and Privacy," pages 120 –132.

Leland, W., Taqqu, M., Willinger, W., and Wilson, D. (**1994**). "On the self-similar nature of Ethernet traffic (extended version)." *Networking, IEEE/ACM Transactions on*, 2(1): 1–15.

Levenshtein, V. (**1966**). "Binary codes capable of correcting deletions, insertions, and reversals." In "Soviet Physics Doklady," volume 10, pages 707–710.

Levine, J., LaBella, R., Owen, H., Contis, D., and Culver, B. (**2003**). "The use of honeynets to detect exploited systems across large enterprise networks." In "Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society," pages 92–99. IEEE.

Li, M. and Vitanyi, P. (**2008**). *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag New York Inc.

Li, Z., Sanghi, M., Chen, Y., Kao, M., and Chavez, B. (**2006**). "Hamsa: Fast signature generation for zero-day polymorphicworms with provable attack resilience." *IEEE Computer Society*.

Liao, H., Tung, K., Richard Lin, C., and Lin, Y. (**2012**). "Intrusion Detection System: A Comprehensive Review." *Journal of Network and Computer Applications*.

Liao, Y. and Vemuri, V. (**2002**). "Use of K-Nearest Neighbor classifier for intrusion detection." *Computers & Security*, 21(5): 439 – 448. doi:10.1016/S0167-4048(02)00514-X.

Lindqvist, U. and Jonsson, E. (**1997**). "How to systematically classify computer security intrusions." In "Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on," pages 154–163. IEEE.

Lindqvist, U. and Porras, P. (**1999**). "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)." In "Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on," pages 146–161. IEEE.

Lipson, H. F. (2002). "Tracking and tracing cyber-attacks: Technical challenges and global policy issues." Technical report, Defense Technical Information Center (DTIC) Document.

Liu, Z., Florez, G., and Bridges, S. (2002). "A comparison of input representations in neural networks: a case study in intrusion detection." In "Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on," volume 2, pages 1708 –1713.

Lough, D. L. (2001). *A taxonomy of computer attacks with applications to wireless networks*. Ph.D. thesis, Electrical and Computer Engineering.

Lunt, T. (1993). "A survey of intrusion detection techniques." *Computers & Security*, 12(4): 405–418.

Mahoney, M. V. and Chan, P. K. (2002). "Learning nonstationary models of normal network traffic for detecting novel attacks." In "Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining," pages 376–385. ACM.

Marcinkowski, S. (2001). "Extranets: The Weakest Link and Security." Technical report, SANS Institute. Retrieved from `http://www.sans.org/reading_room/whitepapers/basics/extranets-weakest-link-security_432`.

McHugh, J. (2000). "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory." *ACM Trans. Inf. Syst. Secur.*, 3(4): 262–294. doi:10.1145/382912.382923.

Mead, N. R. and Stehney, T. (2005). *Security quality requirements engineering (SQUARE) methodology*, volume 30. ACM.

Miller, Z., Dietrick, W., and Hu, W. (2011). "Anomalous Network Packet Detection Using Data Stream Mining." *Journal of Information Security*, 2(4): 158–168.

Miller, Z. and Hu, W. (2012). "Data Stream Subspace Clustering for Anomalous Network Packet Detection." *Journal of Information Security*, 3(3): 215–223.

MITRE (2012). "Common vulnerabilities and exposures (CVE)." Retrieved from `http://cve.mitre.org/`. Accessed on: 05-10-2010.

Mohammed, M., Chan, H., and Ventura, N. (2008). "Honeycyber: Automated signature generation for zero-day polymorphic worms." In "Military Communications Conference," pages 1–6. IEEE.

Moore, A. P., Ellison, R. J., and Linger, R. C. (2001). "Attack modeling for information security and survivability." Technical report, Defense Technical Information Center (DTIC) Document.

Navarro, G. (2001). "A guided tour to approximate string matching." *ACM Comput. Surv.*, 33: 31–88. doi:http://doi.acm.org/10.1145/375360.375365.

Neumann, P. G. (1994). *Computer-related risks.* Addison-Wesley Professional.

Neumann, P. G. and Parker, D. B. (1989). "A summary of computer misuse techniques." In "Proceedings of the 12th National Computer Security Conference," pages 396–407.

Newsome, J., Karp, B., and Song, D. (2005). "Polygraph: Automatically generating signatures for polymorphic worms." In "Security and Privacy, 2005 IEEE Symposium on," pages 226–241. IEEE.

Ning, P. and Xu, D. (2003). "Learning attack strategies from intrusion alerts." In "Proceedings of the 10th ACM conference on Computer and communications security," pages 200–209. ACM.

Ninja, D. (2007). "FUZZY CLARITY: Using Fuzzy Hashing Techniques to Identify Malicious Code1." Retrieved from `http://www.shadowserver.org/wiki/uploads/Information/FuzzyHashing.pdf`.

Northcutt, S. and Aldrich, T. (1998). "SHADOW."

Novak, J. and Northcutt, S. (2003). *Network Intrusion Detection.* New Riders Publishing.

Oikarinen, J. and Reed, D. (1993). "Internet Relay Chat Protocol." RFC 1495.

on National Security Systems, C. (2003). "National Information Assurance Glossary." *Formerly NSTISSI*, 4009. Retrieved from `http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf`.

Padmanabhan, V. and Subramanian, L. (2001). "Determining the geographic location of Internet hosts." *PERFORMANCE EVALUATION REVIEW*, 29(1): 324–325.

Parker, D. B. (1989). "Computer Crime: Criminal Justice Resource Manual ." Technical report.

Paxson, V. (1999). "Bro: a system for detecting network intruders in real-time." *Computer networks*, 31(23): 2435–2463.

Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W. (**2009**). "McPAD: A multiple classifier system for accurate payload-based anomaly detection." *Computer Networks*, 53(6): 864–881.

Phoha, V. V. (**2002**). *Internet security dictionary.* Springer.

Porras, P. and Neumann, P. (**1997**a). "EMERALD: Event monitoring enabling response to anomalous live disturbances." In "Proceedings of the 20th national information systems security conference," pages 353–365.

Porras, P. A. and Kemmerer, R. A. (**1992**). "Penetration state transition analysis: A rule-based intrusion detection approach." In "Computer Security Applications Conference, Eighth Annual Proceedings.", pages 220–229. IEEE.

Porras, P. A. and Neumann, P. G. (**1997**b). "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances." In "1997 National Information Systems Security Conference," Retrieved from `http://www.csl.sri.com/papers/emerald-niss97/`.

Portnoy, L., Eskin, E., and Stolfo, S. (**2001**). "Intrusion detection with unlabeled data using clustering." In "In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA)," pages 5–8.

Pouget, F. and Dacier, M. (**2004**). "Honeypot-based forensics." In "AusCERT Asia Pacific Information Technology Security Conference," Citeseer.

Powell, D. and Stroud, R. (**2001**). "MAFTIA: Conceptual Model and Architecture." *Project MAFTIA IST-1999-11583 deliverable D2.*

Project, H. (**2005**). "Honeywall CDROM Roo 3rd Generation Technology." *Honeynet Project & Research Alliance,[Online] Available: http://www. honeynet. org, Last Modified*, 17.

Project, T. H. (2008, February). "Know Your Enemy: Sebek, A kernel based data capture tool,." `http://www.honeynet.org`, Accessed on: 09-04-2009.

Provos, N. and Holz, T. (**2007**). *Virtual honeypots: from botnet tracking to intrusion detection.* Addison-Wesley Professional.

Ptacek, T. (**1998**). "Insertion, evasion, and denial of service: Eluding network intrusion detection." Technical report, DTIC Document.

Qing, S., Jiang, J., Ma, H., Wen, W., and Liu, X. (**2004**). "Research on intrusion detection techniques: a survey." *China institute of communicateions*, 25(7): 19–29.

Rehak, M., Pechoucek, M., Grill, M., Stiborek, J., Bartos, K., and Celeda, P. (**2009**). "Adaptive multiagent system for network traffic monitoring." *Intelligent Systems, IEEE*, 24(3): 16–25.

Roesch, M., Kane, S., Guiterman, M., et al. (**2006**). "Snort, the de facto standard for intrusion detection." Retrieved from `http://www.snort.org`. Accessed on: 02-06-2010.

Roesch, M. et al. (**1999**). "Snort-lightweight intrusion detection for networks." In "Proceedings of the 13th USENIX conference on System administration," pages 229–238. Seattle, Washington.

Sabahi, F. and Movaghar, A. (**2008**). "Intrusion Detection: A Survey." In "Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on," pages 23 –26.

Sebring, M., Shellhouse, E., Hanna, M., and Whitehurst, R. (**1988**). "Expert systems in intrusion detection: A case study." In "Proceedings of the 11th National Computer Security Conference," volume 32.

Shannon, C. E. (**2001**). "A mathematical theory of communication." *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1): 3–55. doi:10.1145/584091.584093.

Shirey, R. (**2000**). "RFC 2828: Internet security glossary." *The Internet Society.*

Shuja, F. (**2008, February**). "Virtual Honeynet: Deploying Honeywall using VMware." `http://www.honeynet.pk/honeywall/index.htm`, Accessed on: 09-01-2009.

Singh, S., Estan, C., Varghese, G., and Savage, S. (**2004**). "Automated worm fingerprinting." In "Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation," volume 6, pages 4–4. USENIX Association.

Smaha, S. (**1988**). "Haystack: an intrusion detection system." In "Aerospace Computer Security Applications Conference, 1988., Fourth," pages 37 –44.

Smalley, S., Vance, C., and Salamon, W. (**2001**). "Implementing SELinux as a Linux security module." *NAI Labs Report*, 1: 43.

Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C.-L., Levitt, K. N., Mukherjee, B., Smaha, S. E., Grance, T., et al. (**1991**). "DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype." In "Proceedings of the 14th National Computer Security Conference," pages 167–176.

Soft, S. (**1995**). "SunSHIELD Basic Security Module Guide." Accessed on: 23-09-2009.

Sommer, R. and Paxson, V. (**2003**). "Enhancing byte-level network intrusion detection signatures with context." In "Proceedings of the 10th ACM conference on Computer and communications security," pages 262–271. ACM.

Sommers, J., Yegneswaran, V., Barford, P., et al. (**2004**). "A framework for malicious workload generation." In "Internet Measurement Conference: Proceedings of the 4 th ACM SIGCOMM conference on Internet measurement," volume 25, pages 82–87.

Soule, A., Salamatian, K., and Taft, N. (**2005**). "Combining filtering and statistical methods for anomaly detection." In "Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement," IMC '05, pages 31–31. USENIX Association, Berkeley, CA, USA. Retrieved from `http://dl.acm.org/citation.cfm?id=1251086.1251117`.

Spitzner, L. (**2002**). *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Staniford, S., Hoagland, J. A., and McAlerney, J. M. (**2002**). "Practical automated detection of stealthy portscans." *Journal of Computer Security*, 10(1/2): 105–136.

Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D. (**1996**). "GrIDS-a graph based intrusion detection system for large networks." In "Proceedings of the 19th national information systems security conference," volume 1, pages 361–370. Baltimore.

Sung, A. and Mukkamala, S. (**2003**). "Identifying important features for intrusion detection using support vector machines and neural networks." In "Symposium on Applications and the Internet," pages 209 – 216.

Symantec (**2011**). "Symantec Internet Security Threat Report 2011 Trends." *Symantec Internet Security Thread Report*. Retrieved from `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf`.

Tang, Y. and Chen, S. (**2007**). "An automated signature-based approach against polymorphic internet worms." *IEEE Transactions on Parallel and Distributed Systems*, pages 879–892.

Tridgell, A. (**2002**). "Spamsum." Retrieved from `http://www.samba.org/ftp/unpacked/junkcode/spamsum/README`. Accessed on: 08-11-2009.

Vandoorselaere, Y. (**1998**). "Prelude-IDS." Accessed on: 08-09-2009.

Vigna, G. and Kemmerer, R. (**1998**). "NetSTAT: A network-based intrusion detection approach." In "Computer Security Applications Conference, 1998. Proceedings. 14th Annual," pages 25–34. IEEE.

Wang, K. and Stolfo, S. (**2004**). "Anomalous payload-based network intrusion detection." In "Recent Advances in Intrusion Detection," pages 203–222. Springer.

Wang, W., Guan, X., Zhang, X., and Yang, L. (**2006**). "Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data." *Computers & Security*, 25(7): 539 – 550. doi:10.1016/j.cose.2006.05.005.

Wehner, S. (**2007**). "Analyzing worms and network traffic using compression." *Journal of Computer Security*, 15(3): 303–320.

Weiss, G. M. (**2004**). "Mining with rarity: a unifying framework." *SIGKDD Explor. Newsl.*, 6(1): 7–19. doi:10.1145/1007730.1007734.

Werner, T., Fuchs, C., Gerhards-Padilla, E., and Martini, P. (**2009**). "Nebula-generating syntactical network intrusion signatures." In "Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on," pages 31–38. IEEE.

Winters, K., Owsley, P., French, C., Bode, R., and Feeley, P. (1996, July 2). "Adaptive data compression system with systolic string matching logic." US Patent 5,532,693.

Wright, C., Cowan, C., Morris, J., Smalley, S., and Kroah-Hartman, G. (**2002**). "Linux security module framework." In "Ottawa Linux Symposium," volume 8032.

Yegneswaran, V., Giffin, J., Barford, P., and Jha, S. (**2005**). "An architecture for generating semantics-aware signatures." In "Proceedings of the 14th conference on USENIX Security Symposium-Volume 14," pages 1–7. USENIX Association.

Zhang, J., Zulkernine, M., and Haque, A. (**2008**). "Random-Forests-Based Network Intrusion Detection Systems." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(5): 649 –659.

Appendix A

A declaration of previous work

Parts of this thesis are based on previously published materials as follows:

- **Abbasi, Fahim H.**, Harris, R., Marsland, S., and Moretti, G. (2012), "An Exemplar-based Learning Approach for Detection and Classification of Malicious Network Streams in Honeynets". Security Communication Networks Journal, Wiley. Accepted for publication: 15th December, 2012
- **Abbasi, Fahim H.**, Harris R, Morretti G, Haider A, Anwar N. "Classification of malicious network streams in honeynets," Global Telecommunications Conference (GLOBECOM), 2012. Communication and Information System Security Symposium GC12 CISS. IEEE, vol. 12, 2012;3-7 Dec 2012, 6 pp. 35, doi:10.1109/GLOCOM.2012.1578391. URL: `http://www-mobile.ecs.soton.ac.uk/home/conference/Globecom_12/papers/p909-abbasi.pdf`
- **Abbasi, Fahim H.**,Harris, R.J., "Intrusion detection in Honeynets by compression and hashing," Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian , vol., no., pp.96,101, Oct. 31 2010-Nov. 3 2010 doi: 10.1109/ATNAC.2010.5680264 URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5680264&isnumber=5679553`
- **Abbasi, Fahim H.**, and Harris, R.J., "Experiences with a Generation III virtual Honeynet," Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian , vol., no., pp.1,6, 10-12 Nov. 2009 doi: 10.1109/ATNAC.2009.5464785 URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5464785&isnumber=5464714`
- My Citations on Google Scholar: `http://scholar.google.co.nz/citations?hl=en&user=pNpvPl8AAAAJ`