

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/306017096>

A machine learning based approach for intrusion prevention using honeypot interaction patterns as training data

Thesis · May 2016

DOI: 10.13140/RG.2.1.1996.0561

CITATIONS

3

READS

2,127

1 author:



Daniel Zammit

University of Malta

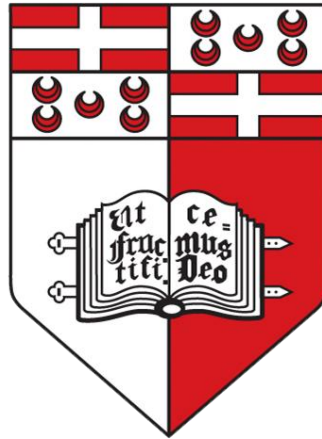
2 PUBLICATIONS 3 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Leveraging Virtual Reality Technology to Effectively Explore 3D Graphs [View project](#)



University of Malta

A machine learning based approach for intrusion prevention using
honeypot interaction patterns as training data

Daniel Zammit

A dissertation submitted in partial fulfilment of the requirements of the Degree
of Bachelor of Science (Honours) in Business and Information
Technology at the University of Malta

May 2016

Abstract

The work conducted in this dissertation revolves around the study of various intrusion detection systems and techniques that are used for detection. Subsequently, a prototype is developed having supervised machine learning capabilities that can be deployed on a network and used by experts to help prevent attacks. The benefits of such an approach is the ability for the system to continue learning with the aid of a supervisor, eliminating the need to continuously update databases used by traditional intrusion detection systems.

A platform containing several honeypots was installed on a virtual machine with unrestricted Internet access. The honeypots were used to collect interaction data generated by attackers. Cowrie, a medium interaction honeypot, was chosen for the prototype. Scripts were written to process this data into a recognisable format by WEKA, an open source machine learning software. The classification file generated by this tool is uploaded to a web server and used to present the result in a simple and concise manner.

The intrusion detection prototype was validated by testing several components of the system. Tests targeted the operation of the platform, the data gathering process, the classification output and web interface. The interface hosted on the web server provides the user with real time status of the platform. The result is a functioning intrusion detection system that relies on machine learning techniques to classify traffic generated from honeypot interactions, with its benefits and limitations.

Keywords: Network Intrusion Detection System; Honeypots; Machine Learning; Web Interface

Acknowledgements

My deepest gratitude goes first and foremost to my supervisor and mentor, Dr. Godwin Caruana, without whose help this work would not have been possible. His knowledge and insight on this research area were invaluable for the completion of this project. He provided me with immense encouragement throughout our regular meetings providing continuous feedback and guiding the direction of this work.

I would like to thank my classmates for an unforgettable experience at University. Special thanks go to Andrew Grech who helped me out when the going was tough. I would also like to thank Marco Ochse for his continuous online support during the development of the prototype.

Last but not least, I would like to thank my family for their tremendous support and encouragement during these years but especially while writing this dissertation.

*Reading furnishes the mind only with materials of knowledge.
It is thinking that makes what we read ours.*

John Locke

Declaration of Authenticity

Student's ID: **415594M**

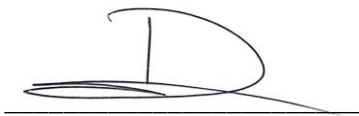
Student's Name and Surname: **Daniel Zammit**

Course: **Bachelor of Science (Honours) Business and Information Technology**

Title of Dissertation: **A machine learning based approach for intrusion prevention using honeypot interaction patterns as training data**

I hereby declare that I am the legitimate author of this Dissertation and that it is my original work. No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education. I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

Word Count: 11765 (excluding headers and tables)

A handwritten signature in dark ink, consisting of a large, stylized 'D' followed by a horizontal line and a small flourish.

Daniel Zammit

May 2016

Table of Contents

Abstract.....	ii
Acknowledgements.....	iii
Declaration of Authenticity	v
List of Tables.....	ix
List of Appendices.....	xi
Chapter 1. Introduction.....	1
1.1 The Necessity of Machine Learning in Network Security.....	1
1.2 Objectives of this research.....	4
Chapter 2. Literature Review.....	6
2.1 Host vs Network Intrusion Detection Systems	7
2.1.1 Host Intrusion Detection Systems.....	7
2.1.2 Network Intrusion Detection Systems	9
2.1.3 Industrial Deployment.....	11
2.2 Honeypot Implementations	12
2.2.1 Honeypot Classification.....	12
2.2.2 Applications	14
2.3 Augmented Signature Based IDS	15
2.3.1 Machine Learning.....	15
2.3.2 Transition from datasets to honeypots.....	16
2.4 Summary.....	17

Chapter 3. Methodology	18
3.1 Design and Prototype Development	18
3.1.1 UML Diagrams	18
3.1.2 Environment Setup	19
3.1.3 Software Development	20
3.2 Testing	21
Chapter 4. Prototype Architecture	24
4.1 Prototype Design	27
4.1.1 Ubuntu Server UML Diagrams	27
4.1.2 Web Host Server UML Diagrams	32
4.2 Prototype Development	34
4.2.1 Hardware Implementation	34
4.2.2 Software Implementation	35
4.2.3 Prototype Functionality	38
4.3 Conceptual Model	44
4.4 Software Testing	45
4.4.1 T-Pot Honeypot Platform	45
4.4.2 Script Data Processing	46
4.4.3 Machine Learning	48
4.4.4 Web Interface	48
Chapter 5. Conclusion and Future Work	50
5.1 Conclusion	50

5.2 Limitations.....	51
5.3 Future Work.....	51
References.....	52

List of Tables

Table 1 - Honeypot levels of interaction	13
Table 2 - Software used during development.....	23
Table 3 - Host Machine Specifications.....	34
Table 4 - Virtual Machine Specifications	35
Table 5 - J48 Pseudocode.....	42

List of Figures

Figure 1 - General System Diagram.....	26
Figure 2 - Ubuntu Server Class Diagram	28
Figure 3 - Ubuntu Server State Machine Diagram.....	31
Figure 4 - Web Server Class Diagram.....	32
Figure 5 - Web Server State Machine Diagram	33
Figure 6 - Kibana Dashboard	36
Figure 7 - Enabling log persistence.....	38
Figure 8 - Retrieving Logs.....	39
Figure 9 - Extracting last entries	39
Figure 10 - Feature Selection.....	40
Figure 11 - Training File	41
Figure 12 - Process Automation using crontab	43
Figure 13 - Prototype OSI	44
Figure 14 - Platform Test Script.....	45
Figure 15 - Cowrie Visualisation on Kibana	46
Figure 16 - Cowrie log entries	47
Figure 17 - Cowrie Machine Learning test file	47
Figure 18 - Classification Result file	48
Figure 19 - Benevolent Traffic.....	49
Figure 20 - Malicious Traffic	49

List of Appendices

Appendix A - Scripts	56
Appendix B - Web Interface.....	72

Chapter 1. Introduction

Business entities in government and private sectors depend on large volumes of information. Millions even billions of data packets enter through the company's network infrastructure and pass from a multitude of devices including tablets, workstations, smartphones, terminals and most importantly servers. A server can be defined as a software which handles a large quantity of queries from the client and is responsible for managing resources such as central processing unit (CPU) time and random access memory between devices connected to it. It can be set up on a single computer for small networks or a cluster of machines known as server racks for large corporations. In the modern world, there was a shift from using servers to delegate workload on machines to what is known as the client-server model which is becoming increasingly popular (Oluwatosin, 2014). The largest network in the world, the Internet, is made of smaller networks that make use of client-server communication. It is used continuously by peers and organisations alike to share information with clients, suppliers and other business partners. Critical information such as client purchase behaviour, transaction history, client personal details and other sensitive information are all stored on servers.

1.1 The Necessity of Machine Learning in Network Security

Servers and databases are primary targets for people who want to access data for malicious purposes. Hackers are people with a unique skillset, capable of infiltrating any system which is vulnerable or exploitable in order to use it for their purposes (Kintana, 2006). There are two types of hackers; ethical and non-ethical hackers, also referred to as white and black hat respectively. An ethical hacker is employed by a company to try and break into their system in order to find means to access that system without triggering security alerts, also known as backdoors, or other unknown weaknesses. On the other hand, a black hat is any person with a good knowledge of hacking tools which they use in order to break into a system without authorisation. Companies spend large amounts of human and financial resources trying to

build defensive mechanisms in order to hold off unauthorised access. Ethical hackers help in the design of such mechanisms. There are various protection methods that can be deployed in a company's internal network, which is known as the intranet, to secure a company's information assets.

As is clear in the literature, in 1983 network security became questionable following attacks targeted at a hospital, bank and a nuclear weapons laboratory in the United States (Elmer-DeWitt et al., 1983). Attackers used home computers to infiltrate and install malware, harmful software designed to compromise a system, which was used to steal user credentials upon the creation of a new user record. The details were immediately sent back to the attacker. Fortunately, the technology that connected us back in the day was still limited to a few machines, which made it simpler to find and apprehend the culprits. Nowadays, networks have grown much bigger than anyone could have imagined back in the early days. The Internet has allowed us to access digitised information created by people and corporations alike. According to Price, "90 percent of all data ever produced by humans has been made in the last two years" (Price, 2015). Nowadays, Internet users and business entities are being targeted everyday by attackers demanding huge sums of money to lift the restriction from the targeted machine. This type of attack is known as ransomware. In a recent study, conducted by PwC and commissioned by the British Minister for Culture and the Digital Economy, Ed Vaizey, a shocking discovery was made: 90 percent of large organisations and 74 percent of small businesses were breached in 2015 (PwC, 2015). With regards to large businesses, 69 percent were infiltrated from outside the company's network. Furthermore, the study also found that distributed denial of service attacks were on the low, following the downward trend forecasted in previous years. Unfortunately, this led to an increase in more sophisticated techniques to hinder daily business operations.

So companies and researchers started to design and develop intrusion detection systems that are capable of detecting misuse or alternations to a system. These systems started to be deployed on both hosts and networks, to analyse certain resources that can be used to

flag unknown malicious behaviour. An example of a network intrusion detection system(NIDS) is Snort, which can be deployed in small or medium sized networks (Roesch, 1999). It is capable of sniffing packets and logging on the network to identify malicious traffic using pattern matching against a defined rule set. On the other hand, a host based intrusion detection system(HIDS), such as OSSEC, is a multi-platform open source monitoring tool capable of performing integrity checks on files, monitor Windows registry and analyse logs in a real time environment on a machine's operating system (Hay et al., 2008). Both OSSEC and Snort are designed to alert a supervisor when a particular pattern or signature matches with a record of known user created signatures of previous attacks, stored in a database. Although such systems are accurate when identifying a known pattern, a problem arises when new attack patterns are used to infiltrate a system. Thus the database needs to be continuously updated. Another technique, known as anomaly based detection does not make use of rules. Instead, the person setting up the system sets a base line for benevolent user behaviour and any deviations from that known user behaviour will be flagged by the system as possibly malicious (García-Teodoro et al., 2009). The disadvantage in anomaly based systems is the high false positive rate. Attackers can also learn how to mimic a normal user's behaviour, thus going under the radar.

In light of this, there is a need for new techniques to capture intruders while mitigating, as much as possible, the disadvantages brought by signature and anomaly based detection systems. Researchers have proposed new techniques that either build on existing solutions or are completely innovative. The literature shows that in a real world environment, companies make use of both host and network based intrusion detection system, and that some research is focused on these hybrid solutions. In the last few years, scientists have started to experiment and propose new methods of applying machine learning techniques to single out intruders in either a host or a network environment. A machine learning aided intrusion detection system has the ability to teach itself over time. The only limit is the amount of data such a system is given to be trained on. Depending on the classification methods utilised and the data inputted for training, the system will be able to classify intruders with more accuracy and less false positives.

1.2 Objectives of this research

The main objective of the dissertation is to propose a system that is capable of capturing and extracting attack data using honeypots. This will be required for the classification of network generated traffic by applying machine learning techniques.

To begin with, the research conducted in this work firstly focuses on the two types of intrusion detection systems mentioned earlier. Host based IDS and network based IDS are reviewed, outlining the differences between them. Moreover, many applications and techniques researchers implemented using these systems are discussed, highlighting the main scientific contributions in this field. Also, in the second section of the literature, honeypots are defined. Most work in this field of research has been published in the last decade or so. Various scientific papers are presented, where these mechanisms have proven to be an ideal tool for capturing the fundamental data generated during attacks (Bringer et al., 2012). The last section of the literature review is dedicated to research related with the improvement on existing signature based IDS and an overview of machine learning in this field.

The software selection for the prototype is justified and issues arising from real life deployment in an organisation are discussed. The way the technology used is set up, the number of issues in deployment should be relatively small. This is discussed in more detail in the methodology. A prototype architecture is devised and a prototype system is developed to utilise data generated by honeypots on an exploitable server, in order to be able to create a training file that will be used for machine learning. The proposed system makes use of both new and modified code. Finally the prototype is tested on a home network to ensure it is functional and that all objectives have been met.

To summarise, the objectives of this dissertation are listed below:

1. Research existing intrusion detection systems to understand how they function
2. Identify the tools needed to carry out data collection and extraction on the proposed system
3. Use of the ideal honeypot components in the proposed system
4. Designing proper prototype illustrations using standardised modelling languages such as UML (Unified Modelling Language) diagrams
5. An implementation of the prototype on a home network which can then be extended to an organisation's network.
6. Proper testing of developed prototype followed by a discussion on achieved results

Chapter 2. Literature Review

Intrusion detection systems are designed to detect malicious data sent to a network with the aim of alarming network administrators of suspicious activity. The ARPANet project, founded by the Department of Defence was the first network to be known for the interchange of data between military personnel, governments and other researchers (Denning, 1989). A few years later, researchers adopted the popular TCP/IP protocol and started to form the “network of networks” as termed by many scientific articles. The Internet came to life in the early 1990s when Tim Berners-Lee created the World Wide Web and the network was becoming more available to the general public. As a result, the Internet started to see all forms of activities and network traffic generated by common users, businesses, researchers, scientists and several other entities. Scientists responsible for the upkeep of the network started to worry about the level of protection the Internet had. This is because of the humongous network size of the Internet where volumes of data were being shifted from one corner of the world to the other. There had already been reports of stolen data and attacks on organisation networks which resulted in the loss of huge sums of money (Elmer-DeWitt et al., 1983). One of the most well-known attacks on the global network originated from a computer program designed to infect a system and replicate itself on other systems connected to a network, also known as a worm. There was a significant push in the last ten years for researching better ways to prevent network breaches, especially due to the increasing number of data being generated each year (Walker, 2015). This led to the need to develop more sophisticated and evolved security network measures which make use of statistics, machine learning and pattern signatures. Systems that make use of these properties are known as intrusion detection systems, a passive solution deployed on network infrastructure with the objective of catching malicious behaviour and alerting management about possible threats. The most common types of IDS are anomaly based or signature based. The former solution recognises malicious behaviours as outliers from the norm by using statistical analysis. On the other hand, signature based IDS cross-references attacks on a pre-set definition of an attack type, following a certain pattern of events related to it.

In this paper, the primary focus of the literature is to compare signature based IDS solutions against proposed machine learning aided intrusion detection systems. To begin with, the first section gives an overview of the two types of intrusion systems, host and network based, outlining their advantages and disadvantages as well as recent advancements identified in the literature. The succeeding section follows with the applications and limitations of different honeypot solutions based on their level of interaction. The last section will cover recent literature on proposed machine learning and enhanced signature based IDS, highlighting the advantages and challenges encountered by other researchers.

2.1 Host vs Network Intrusion Detection Systems

2.1.1 Host Intrusion Detection Systems

Host Intrusion Detection Systems (HIDS) represent a single system which observes external or internal unauthorised access, by identifying and gathering session related data. System analysis is accomplished by monitoring system files and registry state, outputting any changes to log files. Moreover, statistical analysis can be conducted on a host IDS to identify discrepancies between normal user behaviour and the unknown user, depending on the behavioural patterns observed from the moment of obtaining authorisation. A warning signal is sent to the administrator or supervisor to alert of a possible intrusion (Letou & Devi, 2013). Most HIDS use a database to monitor alterations on certain file system objects or memory addresses.

2.1.1.1 Types of HIDS

There are four classifications of host intrusion detection systems which are categorised according to what is analysed for intrusion detection (Boer & Pels, 2005). These are *file system monitors*, *log file analysers*, *connection analysers* and *kernel based IDS*. A file system monitor checks the details of a file object, such as ownership, granted permissions and size with previously gathered information on that object. Moreover, file integrity is analysed using checksums such as MD5 that is used to match the computed data hash value with a former hash of the same file. The drawback of using a file based IDS is that it

does not work in real-time, meaning that hackers can leave no trace, increasing the risk of not being detected. However, Boer & Pels did propose a number of solutions to decrease such risks. For instance, the monitor should be configured in a way that the location of highly sensitive files are unreachable by an outsider. During this phase, the checks that are to be conducted on the selected files and paths need to be finalised. In log file analysis, scripts are run real-time on various dumped logs to look out for loglines matching certain keywords, such as “error”. A connection analyser monitors incoming and outgoing traffic on TCP, UDP and ICMP ports. It listens to these ports and is able to detect foreign malicious connections. An open source, widely used connection analyser is Snort, which can be set up in three different modes: sniffer, packet logger and network intrusion detection (Mary & Devi, 2013). The limitations of early implementations of host intrusion systems include the detection of individual attacks while they are happening or mostly after they have happened. Furthermore, continuous updating of new found attacks by system administrators onto a match or rule database was required. Also, there was no form of intrusion prediction. Coincidentally, these are also some of the disadvantages of a signature based IDS.

2.1.1.2 Modern solutions

The most recent host intrusion detection solutions detect intruders based on deviations from a learned user or system profile, alerting a supervisor with a percentage certainty of an attack. Profiles are generated by analysing behavioural attributes of users using the system over a period of time, gathering a number of characteristics that differentiate from one profile to another (Vokorokos & Baláž, 2010). The researchers highlight the possibility of attackers to assimilate behaviour identical or similar to standard user behaviour profiles. It would be really interesting to find a counter measure for this drawback. These solutions are known as statistical anomaly based detection systems. Such systems use training and testing techniques to be able to correctly classify an intrusion based on theorems or algorithms. The intrusion detection solution proposed by Altwaijry illustrates the use of a naïve Bayesian filter, comprised of a training and testing engine, to calculate the probability of attributes occurring in normal and attack traffic. (Altwaijry, 2011) Other techniques include the use of genetic algorithms to predict an attack or normal type of data. The process is split up into the

training phase, where chromosome groups are generated based on training data and the detection phase, which generates testing chromosome groups. The training and testing chromosome groups are compared after undergoing crossover and mutation processes. The prediction is classified when the final chromosome of test data falls closely under one of the chromosome groups, representing either a normal or attack type (Hoque et al., 2012).

2.1.2 Network Intrusion Detection Systems

On the other hand, Network Intrusion Detection Systems (NIDS) operate at the gateway level, monitoring all incoming traffic both externally and internally on a network. The system generally analyses packet headers against known signatures to either alert the network administrator or suspend all network traffic, depending on the severity of the detected attack (Yadav & Singh, 2013). An early concept of designing an intrusion detection system capable of monitoring network traffic was conceived from applying some known principles of host-based IDS onto a network. The research conducted by Heberlein resulted in a network security monitor (NSM) capable of detecting most attacks targeted at a localised network (Heberlein et al., 1990). NSM would detect an attack based on previously recorded resource usage under normal user behaviour against current, real time resource usage. If usage didn't match with normal user behaviour, it would alert the researcher through the monitor. Six years later, a company called Internet Security System Inc. released a commercialised network intrusion detector named RealSecure, on the Windows NT 4.0 platform (Khandagale & Kalshetty, 2013). Modern NIDS have a packet sniffing module to constantly monitor all packets, focusing more on those that can lead to malicious events. Furthermore, a server is responsible of managing and analysing gathered traffic data while an administrator oversees the whole system (Stallings et al., 2014).

2.1.2.1 Sensor Placement

In a NIDS, it all comes down to the placement of sensors that will be responsible for detecting any malicious patterns in the data being transmitted on the network. The challenge is to lower processing overheads, costs and memory usage. In order to do so, sensors need

to be placed in key positions in the network for the most effect and possibly with the least impact on the network as well as the users. An attack graph technique has the ability to identify paths and locations that attackers are more likely to pass through than others. The graph is produced by taking into consideration the network configuration and placements of critical files. Placing sensors on paths which lead to critical files and therefore more likely to be chosen by attackers, results in a lower cost of sensor deployment as they are only installed where needed, and not throughout the whole network (Noel & Jajodia, 2007). There are two types of sensors which can be deployed: inline and passive sensors.

Inline Sensors

Inline sensors are usually installed with other network devices such as a gateway or at a firewall. The advantage for choosing this alternative is the sensor's ability to stop network traffic the moment it detects an attack (Zou & Chakrabarty, 2003). The disadvantage to this approach is the delay which results from traffic that needs to pass first through the gateway or switch and then through the sensor. Another drawback is the termination of connectivity if an attack is detected. Operations are halted which ultimately lead to downtime costs. The worst case scenario would be when the detector stops all communication because of a false positive. Therefore, most companies use what is known as the passive sensor.

Passive Sensors

The role of a passive sensor is to monitor a copy of the traffic data, so the real traffic never passes through it. These sensors are usually deployed in key network sections, for instance services exposed to the Internet inside a subnet, known as the demilitarised zone. A spanning port which can be found on a switch can be used to analyse traffic circling the network. It is rather cost effective and simple to set up. Problems can occur if during the process the switch is set up incorrectly, which can stop some traffic from entering the sensor. Network tapping and IDS load balancing are some of the other methods to establish a passive sensor. The former utilises the main physical device such as a fibre optic cable to get a copy of network traffic. The downside to this solution is the additional costs needed to buy these attachments and the network downtime. Load balancing also utilises a copy of

network traffic data but in a collected format. It has the ability to redistribute the traffic to multiple sensors across the network based on predefined rules set by an administrator (Kabila, 2008).

2.1.2.2 Enhanced NIDS

Until recently, the literature shows that most of the proposed work on these systems are passive implementations, meaning that no action is taken by the system itself other than to alert the system administrator. Network intrusion detection and prevention systems (NIDPS), an improved system with active defensive capabilities, can be set up to take immediate action, specified by a network administrator at the implementation stage, when the attack is launched rapidly. (Korčák et al., 2014). The decision taken by the NIDPS is not only limited to IP address matching and traffic anomaly detection but also based on port matching via TCP and UDP channels. Normally, this results in the blacklisting of an IP or closing a port (Scarfone & Mell, 2007). The main difference between host and network based IDS is that the latter checks packets targeted at vulnerable systems in real time against a set of header attack data while a host monitors user activity on a system for any anomalies (Stallings et al., 2014).

2.1.3 Industrial Deployment

As expressed by various experts in the literature, in a real world scenario, the optimal solution for defending an organisation from digital attacks from the inside as well as the outside is to implement a network-based IDS together with a host-based IDS as both systems complement each other. Marinova-Boncheva suggests the use of the two types of intrusion detection systems in a proper business setup (Marinova-Boncheva, 2007). She states that intrusion systems should largely focus on host-based modules, later implementing a network IDS solution on top to strengthen the security against attacks. However, Parande argues that HIDS is restrained by the inability of not detecting an attack in real time, unlike a network based IDS (Parande & Kori, 2015). An approach to enhance HIDS is by utilising machine learning techniques in order to not only decrease the delay of attack identification but also increase the accuracy as well as efficiency by lowering the rate

of false positives and IDS evasion. This can be achieved by collecting attack patterns directly from the source with the use of honeypots.

2.2 Honeypot Implementations

A honeypot, in computer security, can be defined as a device containing a number of exploits, which are attractive to attackers who want to infiltrate an organisation's network for malicious intentions. The first honeypot was known as The Deception Toolkit, developed by Frank Cohen in the late 1990s (Cohen, 1998). In his research, Cohen discusses the use of the software he developed and its effectiveness against automated attacks on a system employing the toolkit. Attackers probing such a system will be presented with multiple vulnerabilities consisting of a high number of deceptions. An important feature of the Deception Toolkit was the ability to alert an administrator of all the attacks against deceptions, providing all the information of the techniques used to attempt to break into the system using a particular service, such as "sendmail". This was a game changing breakthrough for computer and network security. Honeypots gained popularity between 2000 and 2001 when there was a sudden outbreak of worms, which are defined as computer programs that are able to replicate and spread rapidly over a network such as the Internet (Fosnock, 2005). These programs posed a threat to networks all over the world as their main purpose was to increase network traffic and therefore increase latency over the whole network. At the time, there was no means to capture the worm for analysis and therefore honeypots were considered the optimal solution for trapping these dangerous programs (Spitzner, 2002).

2.2.1 Honeypot Classification

There are many different honeypots available nowadays, which can be used for various applications. For this research, honeypots are categorised according to their level of interaction with the attacker. The more data that is needed to be collected for analysis, a higher level of interaction will be required. Table 1 illustrates the benefits and drawbacks that

come with each category, according to the research conducted by Mokube and Adams (Mokube & Adams, 2007).

Level of Interaction	Advantages	Disadvantages	Example
Low	<ul style="list-style-type: none"> • easy to set up and cost effective • require little to no expertise • low risk 	<ul style="list-style-type: none"> • provides limited information on specific attacks • lack a complete feature set 	Honeyd
Medium	<ul style="list-style-type: none"> • offer better simulated services • more difficult for attackers to identify • enables logging of more advanced attacks 	<ul style="list-style-type: none"> • increase in security vulnerability • requires more time to implement and a certain level of expertise 	Kippo
High	<ul style="list-style-type: none"> • no simulation, actual OS used for interaction • ability to log huge amounts of attack data 	<ul style="list-style-type: none"> • complex • time consuming • highest probability of risk 	Honeynet

Table 1 - Honeypot levels of interaction

In the literature, honeypots have been used for quite a number of applications in distinct fields of research. For instance, Portokalidis and Bos designed a system known as Sweetbait that utilises honeypots to capture fast worms for automated analysis and signature generation (Portokalidis & Bos, 2007). Sweetbait then sends out continuously updated signatures to both network and host based IDS/IPS in order to neutralise worms in parts of the Internet.

2.2.2 Applications

Dr. Annamma Abraham and her colleagues proposed a real time intrusion network intrusion system that makes use of honeypots together with both NIDS, NIPS and software tools such as Snort as well as firewalls (Prasad.B et al., 2011). In their system, they show how all these tools can work together to create a system of systems that will overcome the drawbacks present when one of them acts on its own. Honeypots were used to continuously extract log data to detect any irregular behaviour.

Honeypot deception was a technique implemented by Kulhalli and Khot to study the attack patterns of trapped intruders (Kulhalli & Khot, 2014). It is used in such a way that the attacker does not become aware that an attack attempt failed, terminates connections to the network and shifts the session onto the honeypot. All interactions with the honeypot are recorded. IP addresses, attack type and other variables can be observed from the graphical user interface of the proposed system.

In *Network Security Using IDS, IPS & Honeypot*, the researchers transformed the honeypot into a dummy server containing a database full of false information and another database containing logs describing packets sent throughout the network (Malav et al., 2016).

The dummy server will be activated when a user without the necessary permission or an intruder trying to deny services to server clients, classified as a denial-of-service (DoS) attack, tries to infiltrate the main server. The IPS being utilised in this system also checks internal transmissions between clients for malicious packets.

To extend the life of a honeypot, Taylor and Hayatle test a model, using Markov Decision Process, that allows honeypots to decide whether to allow certain illicit instructions to operate on the environment (Taylor & Hayatle, 2013). The authors conducted this research in light of anti-honeypot technologies which were rendering intruder traps ineffective.

Botmasters can instruct an exploited system to attack certain components which monitor the execution of commands. If these components fail to execute, then most likely the system is a honeypot. By using this model, the results show that botmasters will have the advantage

over honeypot techniques. The authors argue that the scope of their model is to help the honeypot supervisor configure the system to respond to botmasters' requests in an optimal strategy.

2.3 Augmented Signature Based IDS

Signature based IDS is an approach for detecting intrusions by matching all incoming data on a host or network to a database containing known signature attacks. The main advantage of such a system is the accuracy of identification when detecting known attacks (García-Teodoro et al., 2009). This results in a low false positive percentage. On the other hand, they are not able to detect attacks that have not been inserted into the database before.

Therefore, signature based detection systems, on their own, require continuous updates to keep effective against novel attacks, very similar to how an anti-virus software works.

Advances have been made to improve the reliability and accuracy of these systems. Gupta et al. dedicated their research to discuss the various pattern matching algorithms available in order to give an idea on the efficient algorithms (Gupta et al., 2014). The main focus, for the purpose of this research, is on machine learning techniques applied to signature based intrusion detection systems. Other methods that improve on the concept of these systems are discussed.

2.3.1 Machine Learning

Machine learning (ML) is the procedure of teaching a computer or device to automate a process by using various algorithms and techniques. Arthur Samuel, an American pioneer in the area of artificial intelligence, defines machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed" (Simon, 2013). There are two types of machine learning: supervised and unsupervised. In supervised learning, the necessary information that will help a machine work with unseen data has already been processed. This is labelled as training data. Unforeseen input is labelled as testing data. Supervised learning can be divided into two categories. Classification uses supervised

learning to test if a data entry falls under the first set or the second set of classified data.

Support vector machines (SVM) is an example of a classification technique used for supervised learning (Kotsiantis, 2007). On the other hand, regression is used for continuous data.

2.3.2 Transition from datasets to honeypots

The literature shows that there is an opportunity for more research to be conducted in this area. Most of the early submissions for machine learning implementations utilise datasets containing a set of data to be audited, having both normal and attack data. The KDD99 dataset is a widely used test bed by many researchers to evaluate the performance of their respective proposed intrusion detection systems (Lippmann et al., 2000). Gangwar et al. utilise this dataset in order to compare their self-learning hybrid intrusion detection model that is comprised of base feature selecting classifiers and a data mining classifier, with other proposed intrusion detection methods (Gangwar & Sahu, 2014). In 2009, a review of intrusion detection using machine learning was undertaken and other popular datasets for intrusion detection were shown to be DARPA1998 and DARPA1999 (Tsai et al., 2009). The review also shows that SVM was gaining popularity as it was the most used in this research area. As discussed before, honeypots were used for other applications such as the attempt to capture worms for analysis to help immunise parts of the Internet by distributing signatures to intrusion detection systems (Portokalidis & Bos, 2007). Honeypots were deployed on a large scale inside an enterprise network in order to identify infected hosts (Uzun, 2014). The machine learning system is trained on models that are formalised by using malware samples collected from the honeypots. In using this approach for detecting infected hosts, the results show a significant decrease in false positives, which is expected since 97 honeypots were used.

Jain et al. show how these technologies can all work in unison to create a system such as the proposed hybrid intrusion detection system proposed in their research (Jain et al., 2011). The implementation uses honeypots to gather log data stored in a database to be used as an input for WEKA, a machine learning tool, to generate a real time rule set for Snort, based

on signatures and anomaly techniques, which is capable of classifying normal and malicious traffic.

2.4 Summary

With the increase in traffic that is being generated daily, gigabytes or even terabytes of data are flowing through organisation's networks. Volumes of data located in data repositories are being targeted by assailants to be used for malicious purposes. There is a need for increased security on these networks since in the last few years, multiple cyber-attacks against companies have been reported (Koch et al., 2012). Although there are solutions which do implement either one of the mentioned technologies in the literature, it is either used for other applications or utilised in a different approach. By combining honeypots to gather attack information from an intruder and supervised machine learning techniques it would be possible to create a dynamic training file for an IDS which updates regularly when unclassified data containing new attack features is not registered. The defensive system is therefore able to learn about new threats, allow a supervisor to suggest the action that should be taken and detect existing ones. Furthermore, such a system helps to mitigate the possibility of an intruder teaching the system to consider their attacks as normal data.

The purpose of this work is to research these technologies to contribute academically by developing a prototype that is capable of capturing, monitoring and analysing attacker behaviour in order to extract features from the attacks that can help an intrusion detection system classify them with the help of supervised machine learning. The literature shows that there is room for more research in this particular field. By using machine learning, the signature file that is normally used by some kinds of traditional IDS is replaced by a training file that will be constantly updated when novel attacks are detected or alterations of existing attacks are discovered.

Chapter 3. Methodology

In this section, the approach taken for designing and developing a prototype system that is able to extract data from honeypot logs is discussed. The selection process for honeypots is described and reasons as to why the chosen honeypot was selected are given. The picked honeypot is then used as a stepping stone for the development of the prototype. The implementation of machine learning techniques, using the extracted data, will be subsequently explained.

3.1 Design and Prototype Development

The initial step for developing a prototype is to illustrate how it works on paper, by using diagrams and technical graphical representations. A well-known and accepted standard that is used to design the architecture is explained below. The prototype developed was based on Ubuntu server, running inside a virtualised environment on a host machine, with the honeypots running on this server.

3.1.1 UML Diagrams

The use of Unified Modelling Language (UML) allows for a standardised way of communicating early and finalised concepts of software projects. It is widely used in the field of software engineering and computer science, to show the different classes as well as components. Variables, functions and relationships are depicted in these illustrations in such a way that they are understood by peers and other interested readers. The diagrams were created using *Microsoft Visio 2016* on Windows 10 Home Edition (64-bit version). Two types of UML diagrams are used which are:

- i) Class (Descriptive)
- ii) State (Behavioural)

In addition, a general system diagram was drawn to illustrate the setup of the prototype, other entities and boundaries.

3.1.2 Environment Setup

3.1.2.1 Host Configuration

The prototype built was software based. Software is the intangible part of a computer system. It can be categorised into application software and system software. The latter can be either operating systems or other programs capable of running application software. Meanwhile, application software can be any program that provides functionality to the user. Given the nature of the proposed system, the prototype was built and run in a virtualised environment on a host machine to avoid any security risks.

To set up a virtual environment inside the host, a virtual machine (VM) needs to be created. A virtual machine is a software which runs applications inside a chosen operating system (VMWARE, 2014). It utilises the host physical hardware and resources to be able to operate as well as interact with the user. In order to do this, specialised software was required to be installed. The idea is to run an operating system on top of the host OS. In this case, a Linux operating system will be running on top of Windows 10, inside the virtual environment.

3.1.2.2 Test Environment

The software selected to create this environment was VirtualBox, a virtualisation program developed by Oracle. VirtualBox was chosen mainly for its simplicity, compatibility and active community development. In order to properly utilise this software, these features have to be

enabled on the host: hyper threading and hardware virtualisation. Hyper threading is the process of tricking an operating system into thinking it has more cores than it actually has (Intel, 2002). So for instance, the Intel processor on the host machine, has four cores in total, but to the operating system they are seen as eight threads. So each core is doubled, creating advantages such as efficient resource utilisation, increased processor throughput and enhanced performance, for software that makes use of this technology. Hardware virtualisation is the capability of using system components, for instance a hard disk, as a shared storage space between host and any virtual machines residing on the host. (Turban et al., 2007). The machines operate in isolation and a host can have multiple virtual machines utilising the same system resources. For instance, a large server containing many smaller virtual servers that can each operate using their unique operating system. This tremendously increases efficiency and saves cost as the same hardware is being used by more than one system. In a modern computer system, these settings are probably already set by default and no extra work is required. Nonetheless, steps and illustrations are found in the prototype architecture, showing where these features are found and how to enable them.

3.1.2.3 T-Pot Honeypot Platform

The honeypot package chosen is called T-POT, a multi-honeypot platform that started as a project at Deutsche Telekom AG, a German based telecommunications company located in Bonn (Deutsche Telekom AG, 2016). The platform was deployed on the virtual machine that met the requirements shown above. The development of the proposed prototype was conducted on the platform itself, to keep everything in the same place.

3.1.3 Software Development

The development of the prototype was carried out by getting logs from the honeypots and identifying important information from them. The process is known as feature selection. To do this, scripts were written on the platform. There was quite a selection of scripting languages given that the primary function was to get a file and read from it. To keep the system as simple as possible and with minor modifications, two basic scripting languages

were chosen in total. These are bash and Perl, with the former being a command language. On the other hand, Perl is a script programming language used for text processing and other tasks. Perl5 was used in order to prepare the data for training and testing in the machine learning process.

WEKA was chosen for applying machine learning techniques to the extracted data. It is a data mining tool with open source machine learning software developed by the University of Waikato. WEKA was also the simplest to deploy on the Ubuntu server given that it is built using Java.

A front-end web interface was also developed to show results generated from the system, outside the virtual machine. This was done using hypertext mark-up language (HTML) for creating a basic web page, cascade style sheets (CSS) for styling the interface and finally PHP for server side processing. A free web-hosting account was created for this sole purpose.

The primary objective of this prototype was to gather attacker data from honeypot logs and extract the most relevant information. This information can then be used for classification of data using machine learning to alert network administrators in an enterprise about incoming attacks. Thus the system can be seen as an early detection intrusion system that can flag in almost real-time. This limitation is discussed later on.

3.2 Testing

Prototype testing consisted of seeing that the platform was properly configured such that all the data gathering functionality worked, and the web interface represented the right information to the user. The first step was making sure that the honeypots were accessible to the outside world, thus prone to be targeted by attackers. Before starting the tests, it was

made sure that the platform was accessible from the internal home network. Subsequently, ports were opened according to the documentation of T-Pot platform and the system was left for two hours running in the background to see if any activity is recorded. Putty was installed during testing to redirect the virtual machine's local address to the host machine to visually see what is happening on the Ubuntu server. Furthermore, it was used to generate good traffic on the network during the testing phase. mintty, an open-source terminal emulator for Cygwin, is another software installed to transfer files between Windows and Linux for testing and backup purposes.

At a later stage, a script was written to grab logs generated by honeypots from their directory, and copy them to the operating directory of the user. A second script was written to read from a log file and extract certain lines which were used in the following script. Once the relevant lines are filtered, another script tested was used against the extraction and manipulation of pieces of data from the extracted information to output a result in the format required by the machine learning software WEKA. This script was then enhanced for the final version of the prototype.

Testing was carried out throughout the development of the prototype to find what exactly works best to get the desired result, both in terms of the system itself and the web interface. The testing process is centred on the following components of the architecture:

- i) T-Pot Honeypot Platform
- ii) Script data processing
- iii) Machine Learning
- iv) Web Interface

Screenshots were taken during and after testing the finalised version of the prototype. The web interface is also shown in some screenshots. The following table lists all the software that was used throughout the prototype development:

Name	Publisher	Version No.	Use
Microsoft Visio Professional 2016	Microsoft Corporation	16.0.6868.2060	Design of UML Diagrams
T-Pot Honeypot Platform	Deutsche Telekom AG	16.03	Honeypot Implementation and Deployment
Putty	Simon Tatham	0.67.1010.0	Generation of good traffic during testing
mintty	Andy Koppe	2.3.5	Transfer of files required during testing and backup time intervals
Oracle VM VirtualBox	Oracle Corporation	5.0.16	Creating a virtual machine for T-Pot and isolate the host for security reasons
OpenSSH for Windows	Michael Johnson	7.2p1-1	Requirement for creating a link between host and virtual machine
WEKA	University of Waikato	3.6.13	Preparing data to be trained using a classifier and test data to be classified visually
Notepad ++	Don Ho (Senior Software Engineer)	6.9.1	Development of the simple web interface to display results

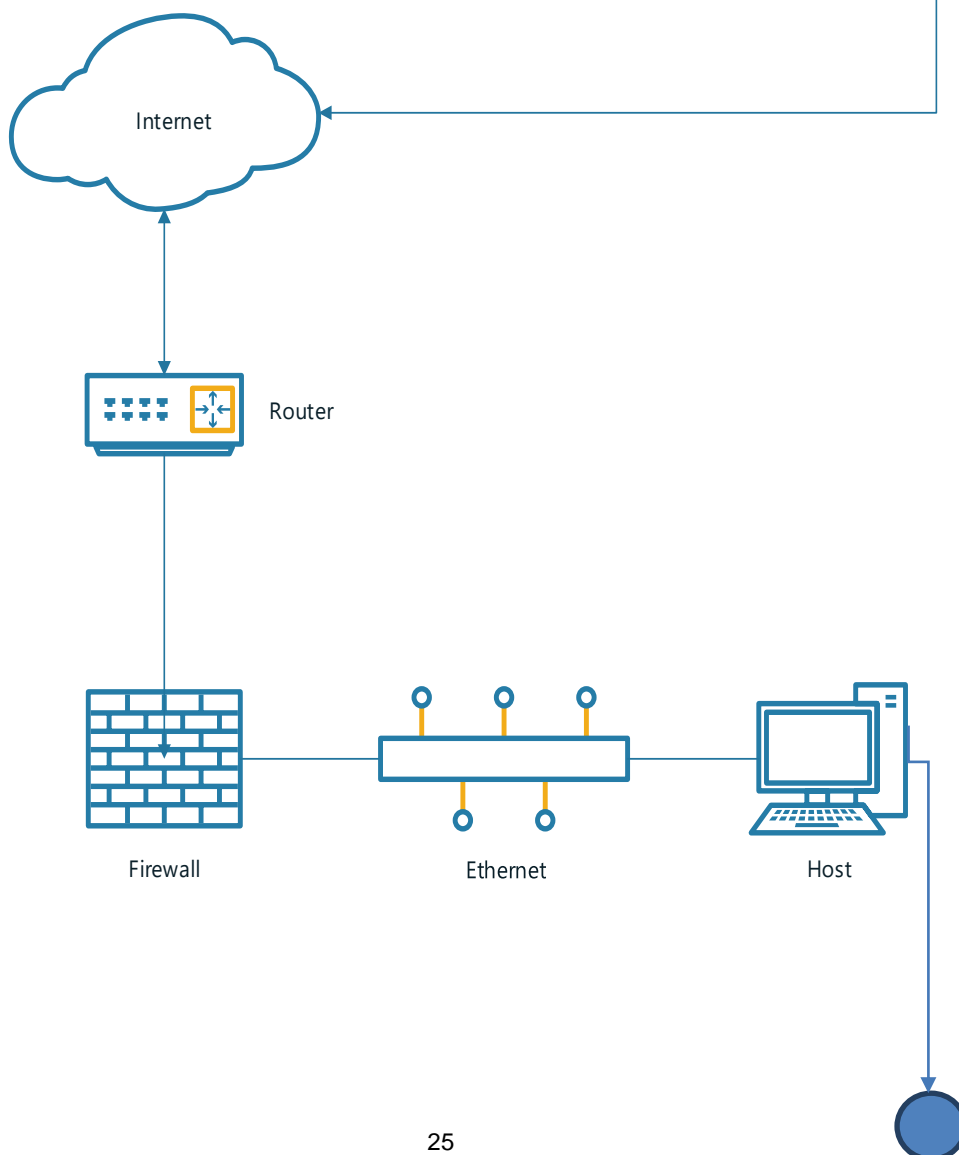
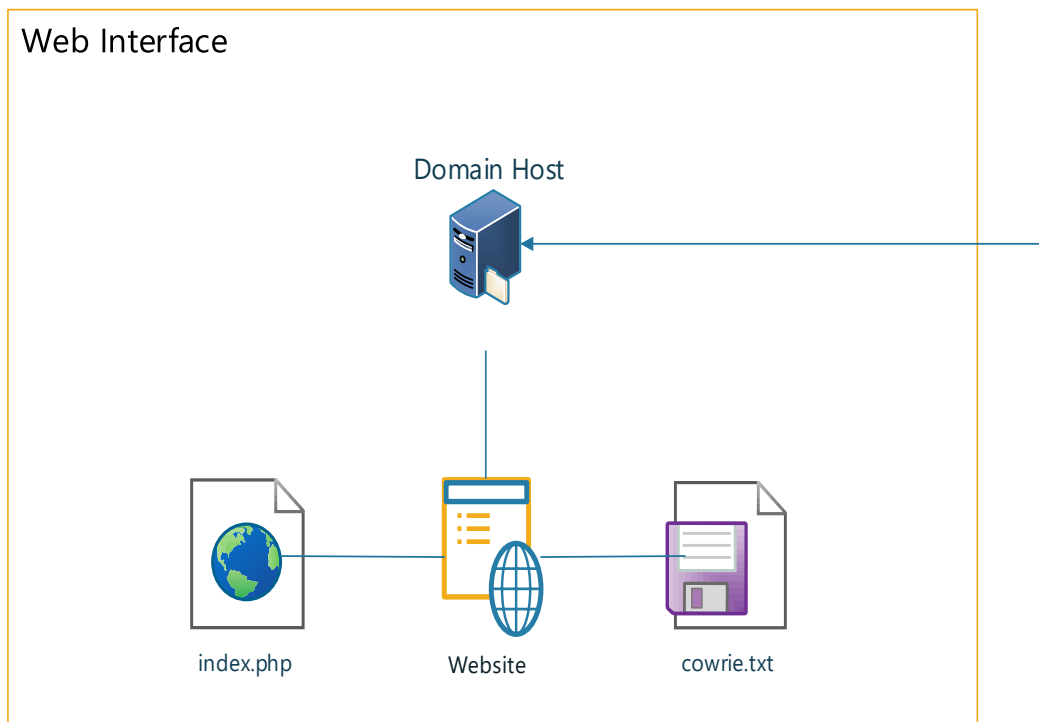
Table 2 - Software used during development

Chapter 4. Prototype Architecture

The prototype built consisted of two components: T-Pot multi-honeypot platform and a web interface. T-Pot was used to deploy several honeypots on a home network, running on an Ubuntu server inside a virtual machine. This was done to isolate the host from the platform. In addition, scripts were written and executed on the platform. Their purpose was to collect, analyse and filter important information from logs generated by honeypots. This was done by implementing pattern matching algorithms, a process known as feature selection. The information selected was then used to build a predictive model using WEKA, an open source collection of data mining tools and machine learning techniques. The last entry in the chosen honeypot log was put against the model to obtain a result. The result was uploaded on a web hosting server for viewing from a network supervisor or any external source. The web interface displayed the result together with the state of the Ubuntu server.

The diagram in the next two pages provides an overview of the system environment and components.

Web Interface



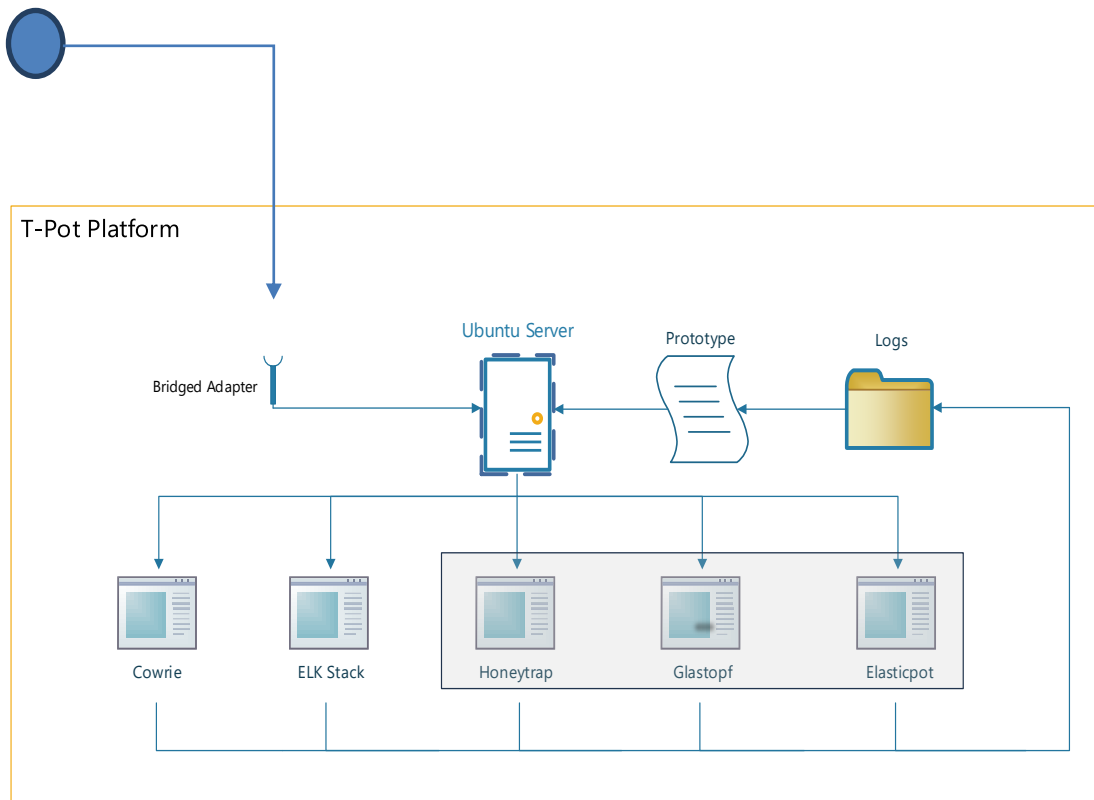


Figure 1 - General System Diagram

The T-Pot honeypot platform operates inside a virtual machine environment on a Windows host machine. The virtual machine's operating system is Ubuntu Server 14.04. The platform contains several honeypots. Cowrie is the honeypot of choice for this work and ELK Stack is used to represent visualisations on the Kibana dashboard. Honeytrap, glastopf and elasticpot are other honeypots that were chosen for future development. Logs of each honeypot are copied to the *Logs* folder, which is then used by the prototype, a combination of scripts that work on these logs and classify traffic accordingly. The virtual machine is connected to the home network via the bridged adapter. Subsequently, this allows the virtual machine to be externally accessed from the Internet and therefore exploitable. Finally, the classification results are transmitted to a web server and can be viewed online.

4.1 Prototype Design

The design phase is an important stage in the prototype development process. It illustrates the main components of the prototype and the relationship between them. As such, UML diagrams have been drawn for the following components:

- i) Ubuntu Server
- ii) Web Host Server

For each component, a class and state machine diagram was drawn. Furthermore, a general system diagram, showing an overview of the whole prototype setup, was also designed.

4.1.1 Ubuntu Server UML Diagrams

The T-Pot platform was deployed inside a virtual machine, with Ubuntu Server as the operating system. The primary goal of the platform is to capture traffic related data that is flagged by the various honeypots available. Scripts that are used to develop the prototype are located on the platform. Also, the classification result of test data is temporarily stored in a file. The platform sends the file to an online web hosting server to display the result on an interface.

4.1.1.1 Class

A UML class diagram is split into two components: class members (attributes or methods) and description section. Class members can have either a “+” notation, signifying a public method or a “-” notation that represents a private method, belonging to that class. The following diagram shows the Ubuntu server in UML class representation:

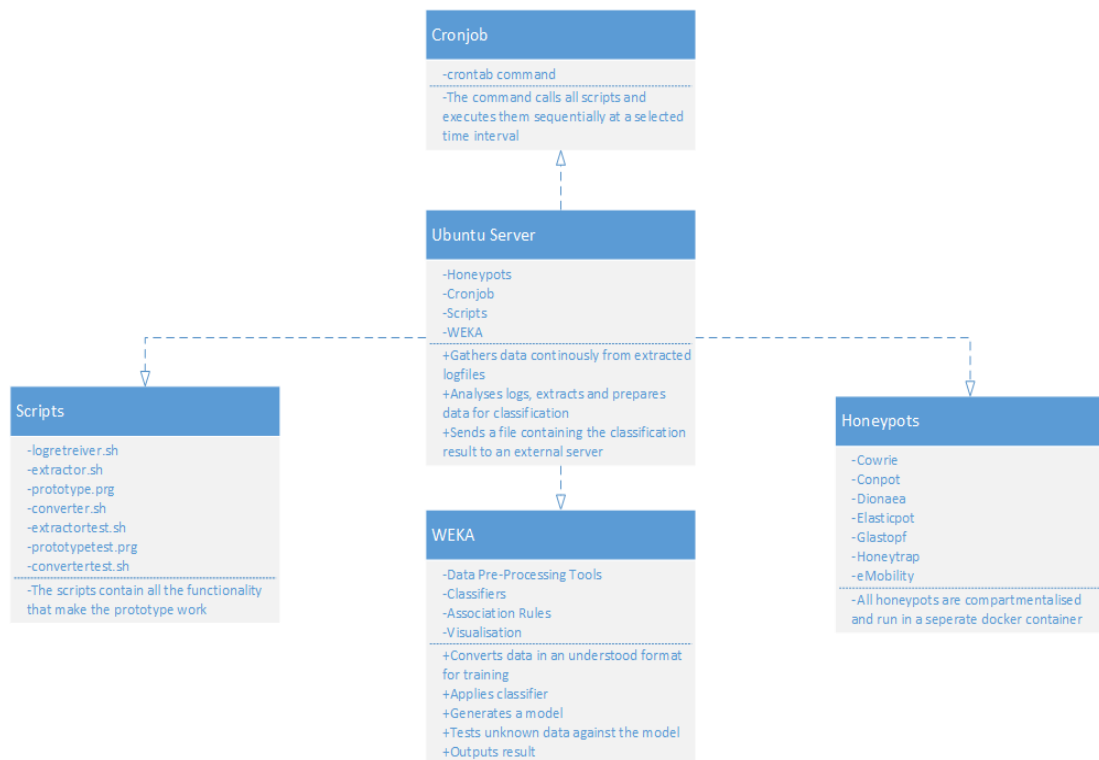


Figure 2 - Ubuntu Server Class Diagram

The diagram represents a grouping of members into four main classes, which are *Honeybots*, *WEKA*, *Scripts*, and *Cronjob*. The latter is the method for automating the prototype process calls every minute. The *Honeybots* class represents an abstract view of honeypots that are found on the T-Pot platform. Each honeypot has its own methods and forwarded port, operating in isolation from other honeypots to avoid conflicts. *WEKA* class contains all the methods for operating WEKA inside the Ubuntu server. The second half of this class shows public methods that can be called externally by scripts and other programs. The *Scripts* class contains all the scripts that are used in the prototype, from the collection, filtering, selection, testing and training of data. Some of these scripts contain private methods and function calls.

4.1.1.2 State Machine

The state diagram at the end of this section illustrates the various states a system can be in at any point in time. The initial state is denoted by a blue circle and the final state by a blue circle with a white outer rim. A state is represented by an edged rectangle. An arrow is used to show a transition from one state to another. This notation can have a trigger, guard or an effect. The trigger symbolises the cause that leads to the transition. A condition must be met in order for the transition to take place. Finally, the effect represents that action taken on the object in control of state machine. This is denoted in the following way: "Trigger [Guard] /Effect"; drawn near the arrow indicating the transition. A state can have multiple states inside of it. These are known as sub-states. On the other hand, the state containing sub-states is known as a composite state. A submachine state denotes complex composite states that need to be separately drawn. The submachine state is illustrated by drawing two smaller states in the bottom right corner. The diagram in the next page shows the states of the prototype.

The initial state of the prototype is the blue circle at the top of the diagram. As soon as the virtual machine running the Ubuntu server has finished loading, the system enters in the *Check Credentials* composite state. The user is shown a details prompt screen asking for username and password. If the details are incorrect, the user is asked to re-enter the details. This process continues until the user is authorised by entering the correct combination of both username and password. As soon as the user logs in, the honeypot services are started. Subsequently, the automated process starts, leading to the crontab. A crontab is a text file containing commands to be executed in a given time, down to minutes, given cron's granularity limitation which is discussed later on (Sharma, 2013). The cronjob, a command that is written to this file, is responsible for executing scripts that put the prototype system in states inside the *Crontab* composite state.

The automated process starts off by retrieving the corresponding logs of each honeypot. The most recent records are then extracted and entries are then filtered. The filtered entries go

through the feature selection process that is used later for machine learning. At this stage, the features that are used for machine learning are selected by text processing methods that make use of pattern matching algorithms written in Perl. The selected features are then converted into a format that is recognised by the machine learning software, WEKA.

Classification (Higher) is a submachine state that represents the machine learning states.

Within this state, the submachine state *Classifying (Higher)* represent the actual training and testing states that are used during the machine learning stage. Training occurs only if no training file is found. For the purposes of this prototype, training was done only once. If the training file already exists, training is skipped and the processed data is tested against the model generated during the training phase. Finally, the result is uploaded onto the web host server and viewed online. This whole process is repeated every minute to continuously check for possible attacks.

If the process is concluded, it waits for the next minute to start. The extra time is spent idling, waiting for the next cycle to start. The *Idle* state can be reached after completing the automated process, starting the honeypot services or logging into the system. The *Off* state can be reached from any state when the user wishes to terminate the Ubuntu server.

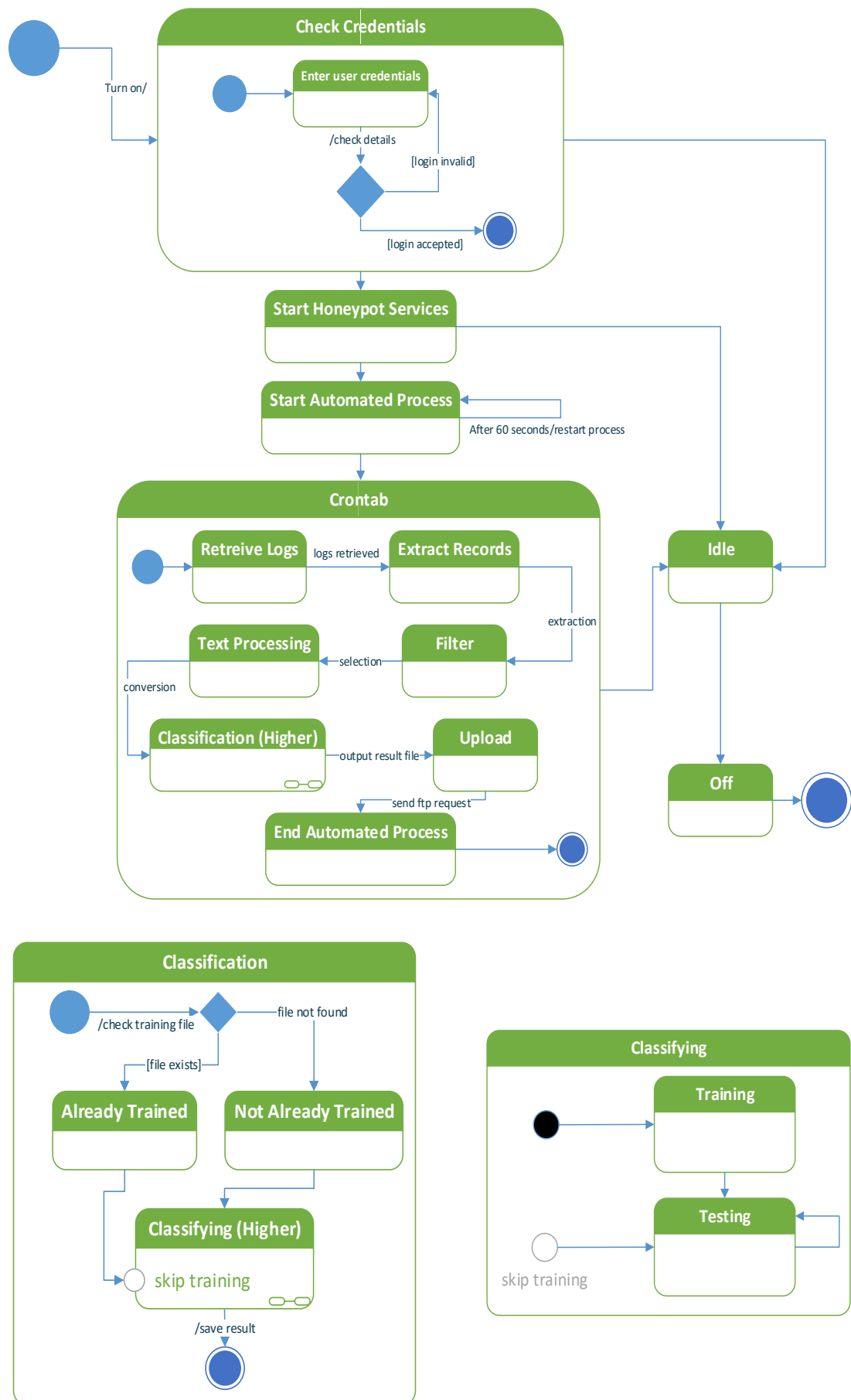


Figure 3 - Ubuntu Server State Machine Diagram

4.1.2 Web Host Server UML Diagrams

The web server receives a request from the Ubuntu server to download the classification result file, at the end of the automated process cycle. This server is used to read from this file, which is used to show the results on a web page together with a status representing whether the Ubuntu server can be reached. The landing page is written in HTML and PHP for file processing and presentation.

4.1.2.1 Class

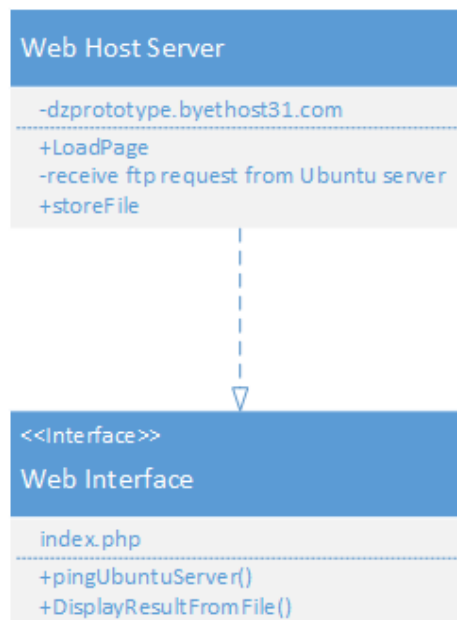


Figure 4 - Web Server Class Diagram

The web server hosts a website titled *dzprototype.byethost31.com*, a free domain provided by Byethost. The site has two important files that contribute to the functionality of the prototype: `index.php`, which is the main page where the result is displayed and the result file. The *Web Host Server* class represents the domain and the main functionality of the server. The main page is loaded every thirty seconds to get the latest result from the Ubuntu server to transfer the most recent classification. The server stores this file for reading from the *Web*

Interface. The interface is basically the main page where the result is shown and the Ubuntu server is sent packets to check whether it is online or offline.

4.1.2.2 State machine

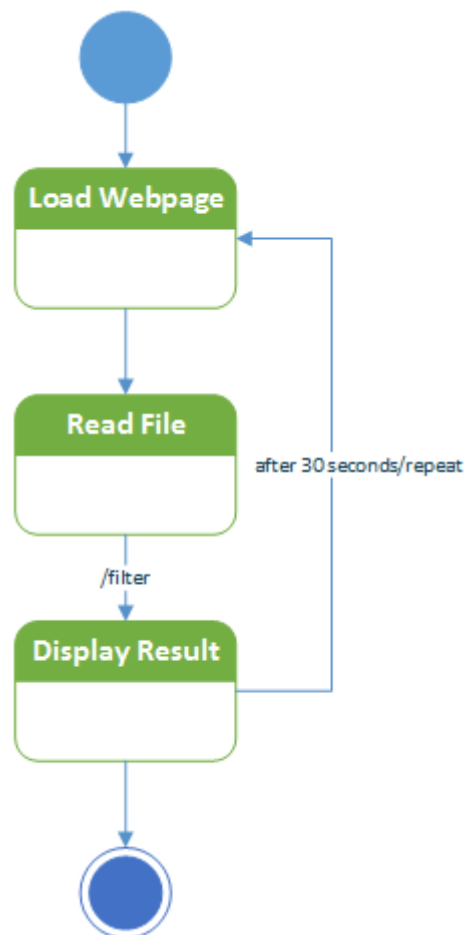


Figure 5 - Web Server State Machine Diagram

The first state of the web server is initialising the main webpage, index.php. Subsequently, the webpage reads from the sent result file located on this server. Then, the text in the result file is filtered and the classification result is presented on the interface. Every thirty seconds, the process is repeated.

4.2 Prototype Development

Following the design of UML diagrams above, a prototype was developed to illustrate how all the classes as well as state transitions of both the platform and web host server can work in practise. The scripts that were used for the prototype development can be found in the appendices.

4.2.1 Hardware Implementation

The prototype makes use of two main physical systems, the host and the web server. The free web server contains all the files for the web interface. On the other hand, the host machine is the system running the virtual machine. The host machine has the following specifications:

Host Machine	
Operating System	Windows 10 (64-bit)
Memory	16GB
Central Processing Unit	Intel Core i7-4790K
Primary Storage Device	Solid State Drive (Capacity 256GB)
Secondary Storage Device	Hard Disk Drive (Capacity 500GB)
Graphics Processing Unit	NVIDIA GTX 970 (Memory 4GB)

Table 3 - Host Machine Specifications

4.2.2 Software Implementation

As previously established, the prototype produced in this work is software. The prototype was developed on a platform that was chosen for its multiple honeypot implementation inside a server. The platform is called T-Pot, and as mentioned earlier it is developed by Deutsche Telekom AG. To deploy the platform, a virtual machine with the following specifications was created, to meet the requirements that are mentioned in the documentation (Deutsche Telekom AG, 2016).

Virtual Machine	
Operating System	Ubuntu Server 14.04.4 LTS (64-bit)
Allocated Memory	6GB
Virtual Central Processing Unit(s) (CPU)	2
Virtual Machine Disk	64GB
Display Settings	Default
Network	Bridged Adapter
Name	Host Ethernet / Wireless Connection Adapter
Adapter Type	Intel Pro/1000 MT Desktop (82540EM)
Promiscuous Mode	Allow VMs
USB	USB 2.0 (EHCI) Controller

Table 4 - Virtual Machine Specifications

The virtual machine was created using VirtualBox. The memory allocated for the machine was increased to 6 gigabytes to ensure stable operation for long periods of time. This was needed as the virtual machine had to be left on for hours in order to collect sufficient attack data for analysis.

4.2.2.1 Virtual and Host Machine Setup

The first step for installing the platform on the virtual machine was to download an image file containing all the files required for installation. The image file, *tpot.iso*, is mounted onto the virtual machine's optical disk drive. Two CPUs were allocated to the virtual machine for greater performance. In addition, network adapter was enabled and set to bridged adapter so that the virtual machine is given Internet access. Also, some ports had to be opened for attackers to target the honeypots. Most importantly, port 64295 was opened to enable remote access through a secure shell protocol, known as SSH. Also, port 22 was opened and port forwarded to allow **cowrie** honeypot to receive traffic.

Furthermore, OpenSSH for Windows was installed, enabling PuTTY to connect with the Ubuntu server inside the virtual machine. PuTTY was used to transfer the platform's local IP address (127.0.0.1) onto the host machine IP address, on port 8080. This was done using the following command: `ssh -l tsec -p 64295 -L8080:127.0.0.1:64296 192.168.0.17`; where `-l` represents the login name, `-p` represents the port, `-L` is the bind address, followed by the port, host and host port. The address at the end is the address of the virtual machine given by the network. To verify that this command works successfully, the following dashboard should come up when entering the local IP address, in a browser, on the host machine:

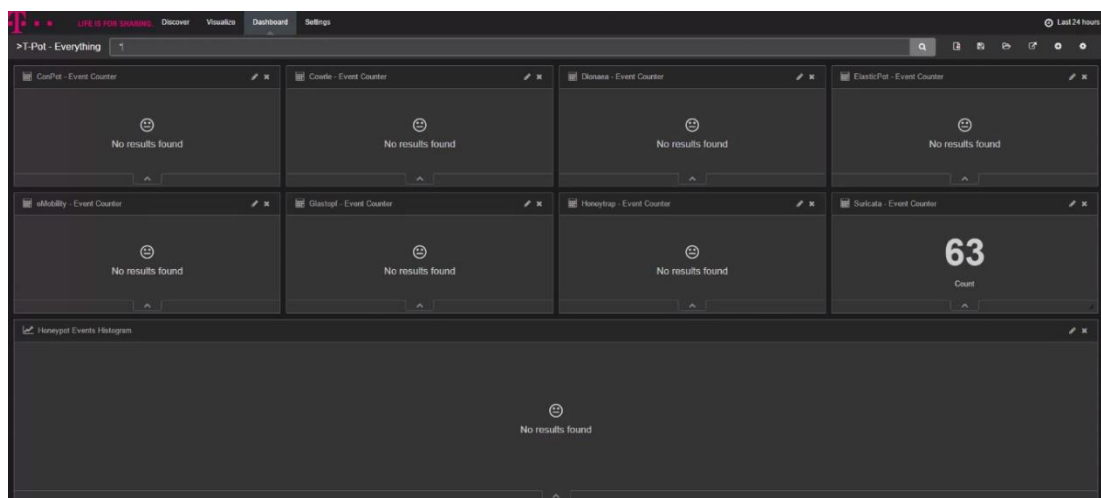


Figure 6 - Kibana Dashboard

In order to get the above result when looking up the local address from the browser, it is best to open port 9200 TCP. Kibana is a visualisation tool for representing historical and real time data that was gathered on the honeypots. It has many dashboards to choose from, from minimalistic and specific to industrial dashboards. In addition, Cygwin, also known as minty was installed to establish a link between the Ubuntu server and Windows to backup files on request, using the following command line:

```
rsync -avzu --exclude='/*/.local' -e 'ssh -p 64295' --progress tsec@192.168.0.17:/home/tsec/*  
/cygdrive/c/Users/Daniel/Desktop/Logfiles/ALLTSEC
```

A file copying tool, called rsync, is used to keep files updated on two different computers. This is done by uploading only the changes in the files. `--avzu` denotes compression techniques while the rest of the command line establishes a secure tunnel using ssh to copy files from Ubuntu to Windows.

4.2.2.2 T-Pot Configuration

The T-Pot platform requires less effort to configure. The most important configuration was done on the root user where the honeypot logs were being stored. A file called *persistence.off* was renamed to *persistence.on*. This was done to keep all the entries of the honeypots after shutting down the machine. New directories were created on the server as seemed fit during development of the prototype. Also, WEKA was installed as it was not pre-installed on the platform.


```

root@cel45804980927713:/home/tsec/prototype/protoscripts# ./logretriever.sh
Retrieving all logs from their directories

-----
Cowrie files successfully retrieved..100%
Dionaea files successfully retrieved..100%
Elasticpot searches retrieved..100%
Glastopf files retrieved..100%
HoneyTrap files retrieved..100%
Suricata files retrieved..100%

Files have been transferred to the secret location

Script run successfully :)
-----
Mon May 23 02:48:43 CEST 2016
root@cel45804980927713:/home/tsec/prototype/protoscripts#

```

Figure 8 - Retrieving Logs

4.2.3.2 Preparing data for machine learning

There are some similar scripts that are used in this work. Due to the nature of the prototype, two scripts were required for each functionality implemented. Similar scripts were created, which were modified for preparing the test data that is used in the machine learning stage.

Training

After retrieving all the logs, the next step is to extract the last entries recorded in each log. In the script *extractor.sh*, this is done by using a command line tool called *grep*. The tool is used to read a file and return matching expressions identical to what is specified in its argument. The *tail* command is used to start reading from the end of file. The results are sorted and unique entries are outputted into a new log file.

```

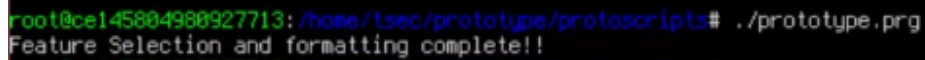
root@cel45804980927713:/home/tsec/prototype/protoscripts# ./extractor.sh
Terminal Out OK!

```

Figure 9 - Extracting last entries

The next step is to identify attributes that will be useful for classifying good or malicious traffic, known as feature selection. For this prototype, traffic classification was done on **cowrie** honeypot logs. Cowrie is a medium interaction honeypot with the purpose of logging brute force attacks and interactions performed by an attacker (Oosterhof, 2015). A script,

prototype.prg, was written to read through the entries extracted by the previous script and format the data in such a way that it can be used in machine learning.

A terminal window with a black background and green text. The prompt is 'root@ce145804980927713: /home/tsec/prototype/protoscripts#'. The command entered is './prototype.prg'. The output is 'Feature Selection and formatting complete!!'.

```
root@ce145804980927713: /home/tsec/prototype/protoscripts# ./prototype.prg
Feature Selection and formatting complete!!
```

Figure 10 - Feature Selection

Finally, the *converter.sh* script takes the formatted attributes and converts them into a format that is understood by WEKA. The result is *cowrie.arff*, a training file, which contains the chosen attributes. At this point, the expert checks each entry in the training file and inputs a 1 for malicious and 0 for non-malicious. This is a very important stage for the machine learning process.

```

@relation cowrie

@attribute Port numeric
@attribute Status {succeeded,failed}
@attribute AttemptOnPort numeric
@attribute AttemptsOnIP numeric
@attribute Malicious {0,1}

@data

1024,succeeded,1,1,0
36881,failed,1,6,0
39229,failed,1,6,0
41250,failed,1,6,0
42797,failed,1,6,0
44417,failed,1,6,0
46253,failed,1,6,0
48794,failed,1,31,1
49075,failed,1,31,1
49345,failed,1,31,1
49524,failed,1,31,1
53192,failed,1,31,1
53260,failed,1,31,1
53321,failed,1,31,1
53400,failed,1,31,1
53444,failed,1,31,1
53479,failed,1,31,1
58788,failed,2,15,1
58788,failed,1,15,1
58790,failed,1,15,1
58790,failed,1,15,1
59515,failed,1,15,1
59515,failed,2,15,1
59515,failed,1,15,1
59529,succeeded,1,2,0
59529,failed,1,15,1
59529,failed,1,15,1
59543,succeeded,1,2,0
59543,failed,3,15,1

```

Figure 11 - Training File

The training file is made up of four attributes in total, apart from the **Malicious** attribute. The first attribute, **Port**, represents the source port of the attacker. **Status** shows if a particular username/password combination was correctly entered. The attacker succeeds if the combination entered is found in the user details text file. **AttemptsOnPort** represents the number of times in succession, an attacker tried to enter on the same port. Most of the time its only once since attackers are smart enough to always try different ports. That is why, on the other hand, **AttemptsOnIP** is the number of times the specific IP of that attacker was found in the 100 entries. If it lies between 3 and 6, then probably it is just a user who forgot their password. Meanwhile, two digit numbers are big enough to indicate a brute force attack. The expert looks at this data and decides whether an entry in the training file, is

malicious or not, by analysing all these properties. The training file is converted into a model to be used at a later stage for testing data.

Testing

The process and functionality for acquiring the data to be tested against the model utilises existing functionality that is already implemented in the training scripts, with some minor tweaks. The logs are retrieved using the same script that was written for the training phase, *logretriever.sh*. In *extractortest.sh*, the `grep` command is used to get the last 60 entries that match an expression instead of the last 100. The *prototypestest.sh* script works on these last 60 entries to generate the attributes mentioned in the training phase. The difference is that the outputted file now contains only one data entry that is tested against the generated model.

A script, *convertertest.sh*, then takes this file, containing one data entry and tests it against the model, using the J48 classifier. This classifier is an open source implementation of the C4.5 algorithm, using Java. The J48 class in WEKA generates a decision tree. The pseudocode which can be seen in Table Y is adopted from a book called *Top Ten Algorithms in Data Mining* (Wu et al., 2008).

J48 Algorithm J48(T)
Input: a feature-valued training dataset T 1. Tree = {} 2. if T is "pure" OR other stopping criteria met then 3. terminate 4. end if 5. for all attribute x contained in T do 6. compute information-theoretic criteria if split on x 7. end for 8. $x_{supreme}$ - Best attribute according to computations above 9. Tree - creates decision node that tests $x_{supreme}$ in root 10. T_w - derived sub-datasets from T based on $x_{supreme}$ 11. for all T_w do 12. $Tree_w = J48(T_w)$ 13. Attach $Tree_w$ to the corresponding branch of Tree 14. end for 15. return Tree

Table 5 - J48 Pseudocode

The J48 algorithm relies on information gain. While traversing the tree, starting from the root node, the paths that offer the most information gain are chosen. Therefore during classification, the algorithm traverses each node to find the highest information gain that can be achieved for the given test data and classifies it accordingly.

4.2.3.3 Process Automation

```

T-Pot 16.03
Hostname: ce145804980927713
IP: 192.168.0.17, 141.8.83.213

T-POT

CTRL+ALT+F2 - Display current container status
CTRL+ALT+F1 - Return to this screen
ce145804980927713 login: tsec
Password:
Last login: Mon May 23 02:09:12 CEST 2016 on tty1
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
No mail.
tsec@ce145804980927713:~$ sudo crontab -l
[sudo] password for tsec:
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
* * * * * /home/tsec/prototype/protoscripts/logretriever.sh && /home/tsec/prototype/protoscripts/extractortest.sh && /home/tsec/prototype/protoscripts/prototypetest.prg && /home/tsec/prototype/protoscripts/convertertest.sh && /root/uploader.sh >> /home/tsec/prototype/debuglogs/final_log 2>&1
tsec@ce145804980927713:~$

```

Figure 12 - Process Automation using crontab

The prototype's purpose is to continuously collect traffic data for classification in real time. With crontab, a tool designed to run commands or scripts at a specific time, a command shown in Figure 11 was written to run the scripts in order every minute. This was done given crontab's one minute granularity limitation.

4.3 Conceptual Model

The following illustration is an adaptation of the open systems interconnection (OSI) model for the developed prototype.

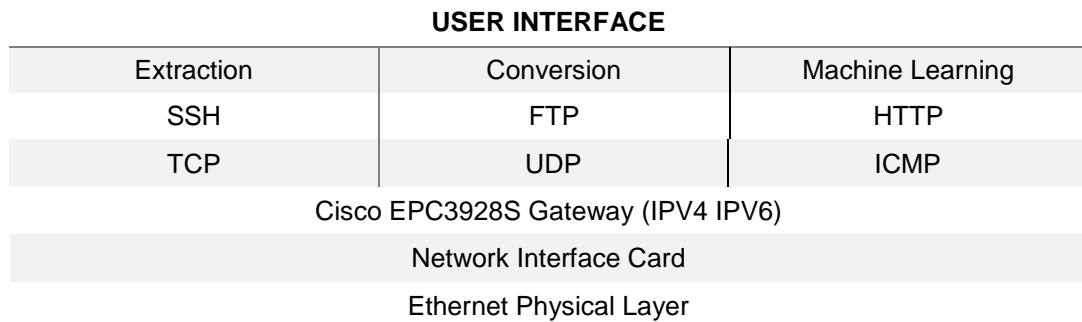


Figure 13 - Prototype OSI

The first layer of the model represents the physical structure that is in place to receive and transmit raw communication data. The *Network Interface Card* sits on the second layer, which is the data link. It provides transfer of data from one node to the other. In this case, the data from the virtual machine to the host machine. The gateway lies on the network layer and is responsible for selecting where packets need to go. The transport layer takes care of traffic control using the three most popular protocols, which are also utilised in the prototype. The session layer contains SSH, FTP and HTTP protocols which are used to manage connections between the host machine, the virtual machine and the web hosting server. In the presentation layer, data is formatted in a way that it can be interpreted and understood by the application layer. The *User Interface* allows a user to view the results of classification.

4.4 Software Testing

Tests were carried out throughout the development stage to check that the platform is working properly and that the scripts function as intended.

4.4.1 T-Pot Honeypot Platform

The platform was tested by checking that the resource load was balanced between the honeypots and that all honeypots were operational during runtime. A script was executed using the following command line to conduct these tests: `sudo status.sh`.

```
===== System =====
Date: Tue May 24 12:40:47 CEST 2016
Uptime: 12:40:47 up 2 min, 1 user, load average: 0.61, 0.58, 0.24
No sensors found!
Make sure you loaded all the kernel drivers you need.
Try sensors-detect to find out which these are.
CPU temp:

===== Container: cowrie =====
cowrie RUNNING pid 7, uptime 0:03:15
euserposter RUNNING pid 9, uptime 0:03:15
mysqld RUNNING pid 8, uptime 0:03:15

===== Container: dionaea =====
dionaea RUNNING pid 9, uptime 0:03:15
euserposter RUNNING pid 8, uptime 0:03:15

===== Container: elasticpot =====
elasticpotpy RUNNING pid 9, uptime 0:03:15

===== Container: elk =====
elasticsearch RUNNING pid 7, uptime 0:03:15
kibana RUNNING pid 13, uptime 0:03:15
logstash RUNNING pid 9, uptime 0:03:15

===== Container: glastopf =====
euserposter RUNNING pid 8, uptime 0:03:15
glastopf RUNNING pid 7, uptime 0:03:15

===== Container: honeytrap =====
euserposter RUNNING pid 9, uptime 0:03:16
honeytrap RUNNING pid 10, uptime 0:03:16

===== Container: suricata =====
p0f RUNNING pid 9, uptime 0:03:16
suricata RUNNING pid 8, uptime 0:03:16

tsec@ce145804980927713: $
```

Figure 14 - Platform Test Script

The script showed the load, the status of each honeypot and their uptime, including that of the platform. Furthermore, to confirm that the platform was connected to the Internet and malicious users could attack it, the Kibana dashboard was used to represent visually the data gathered by honeypots. This was done a week after the honeypot was deployed, to gather a significant amount of data, as shown below:

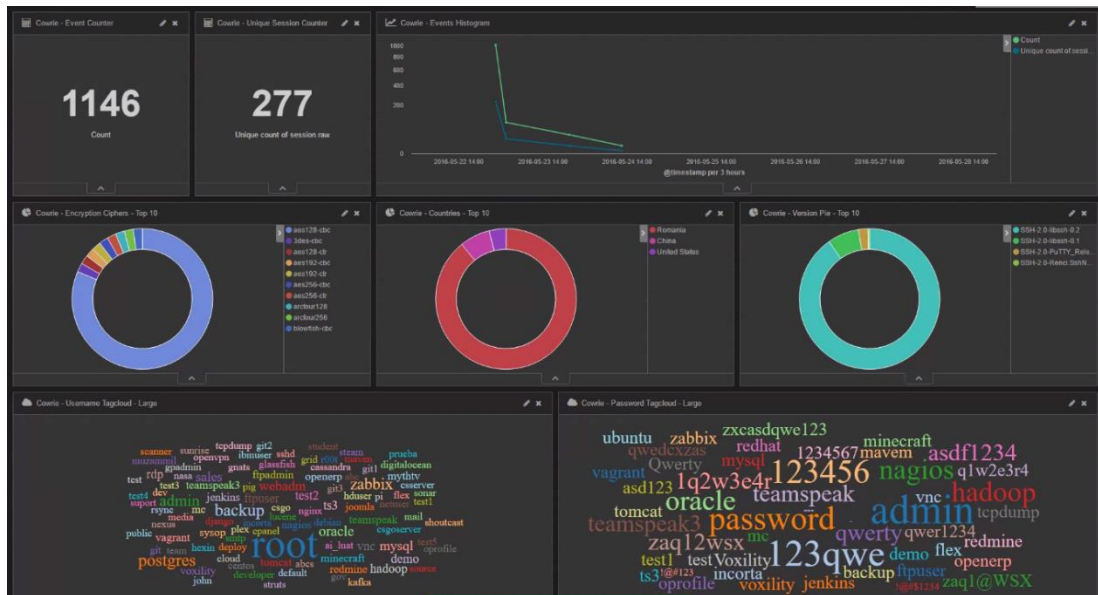


Figure 15 - Cowrie Visualisation on Kibana

The Username and Password Tagcloud shown in Figure 15 represents the different combinations used by attackers to try and infiltrate the Ubuntu server found inside the virtual machine. All connection attempts redirect back to Cowrie for monitoring.

4.4.2 Script Data Processing

The scripts that were executed for processing honeypot data logs created result logs as output, which were used by the next script in the crontab, in the order shown in Figure 12.

The images below show the input data used by *prototypetest.sh* and the outputted data which is then used as an input for the test data classification.

```

GNU nano 2.2.6 File: courieresulttest.log
2016-05-23 00:43:43+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:58490 (172.17.0.2:2222) [session:$
2016-05-23 00:43:44+0000 [SSHSservice ssh-userauth on HoneyPotTransport,239,195.178.182.109] login attempt [root/123ewqasdcxz] failed
2016-05-23 00:43:48+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:59753 (172.17.0.2:2222) [session:$
2016-05-23 00:43:49+0000 [SSHSservice ssh-userauth on HoneyPotTransport,240,195.178.182.109] login attempt [root/321qwedsazxc] failed
2016-05-23 00:43:53+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:32780 (172.17.0.2:2222) [session:$
2016-05-23 00:43:54+0000 [SSHSservice ssh-userauth on HoneyPotTransport,241,195.178.182.109] login attempt [oracle/0racle] failed
2016-05-23 00:43:59+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:34041 (172.17.0.2:2222) [session:$
2016-05-23 00:43:59+0000 [SSHSservice ssh-userauth on HoneyPotTransport,242,195.178.182.109] login attempt [backup/backup18#] failed
2016-05-23 00:44:04+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:35301 (172.17.0.2:2222) [session:$
2016-05-23 00:44:04+0000 [SSHSservice ssh-userauth on HoneyPotTransport,243,195.178.182.109] login attempt [backup/backup123!0#] fail
2016-05-23 00:44:09+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:36550 (172.17.0.2:2222) [session:$
2016-05-23 00:44:10+0000 [SSHSservice ssh-userauth on HoneyPotTransport,244,195.178.182.109] login attempt [backup/backupbackup] fail
2016-05-23 00:44:14+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:37821 (172.17.0.2:2222) [session:$
2016-05-23 00:44:15+0000 [SSHSservice ssh-userauth on HoneyPotTransport,245,195.178.182.109] login attempt [backup/lqazxsw2] failed
2016-05-23 00:44:19+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:39083 (172.17.0.2:2222) [session:$
2016-05-23 00:44:20+0000 [SSHSservice ssh-userauth on HoneyPotTransport,246,195.178.182.109] login attempt [backup/zaq12wsx] failed
2016-05-23 00:44:25+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:40344 (172.17.0.2:2222) [session:$
2016-05-23 00:44:25+0000 [SSHSservice ssh-userauth on HoneyPotTransport,247,195.178.182.109] login attempt [backup/zaq1xsw2] failed
2016-05-23 00:44:30+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:41600 (172.17.0.2:2222) [session:$
2016-05-23 00:44:31+0000 [SSHSservice ssh-userauth on HoneyPotTransport,248,195.178.182.109] login attempt [backup/test123] failed
2016-05-23 00:44:35+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 195.178.182.109:42871 (172.17.0.2:2222) [session:$
2016-05-23 01:10:28+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 13.92.84.29:1040 (172.17.0.2:2222) [session: cbl1$
2016-05-23 01:10:30+0000 [SSHSservice ssh-userauth on HoneyPotTransport,250,13.92.84.29] login attempt [admin/admin] succeeded
2016-05-23 01:43:11+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:44081 (172.17.0.2:2222) [session:$
2016-05-23 01:43:14+0000 [SSHSservice ssh-userauth on HoneyPotTransport,0,218.200.188.213] login attempt [ai.just/pulaaaaaaaaaaaaaa]
2016-05-23 01:43:15+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:44556 (172.17.0.2:2222) [session:$
2016-05-23 01:43:18+0000 [SSHSservice ssh-userauth on HoneyPotTransport,1,218.200.188.213] login attempt [pi/raspberrypi] failed
2016-05-23 01:43:20+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:45335 (172.17.0.2:2222) [session:$
2016-05-23 01:43:22+0000 [SSHSservice ssh-userauth on HoneyPotTransport,2,218.200.188.213] login attempt [hexin/hexin] failed
2016-05-23 01:43:24+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:45902 (172.17.0.2:2222) [session:$
2016-05-23 01:43:26+0000 [SSHSservice ssh-userauth on HoneyPotTransport,3,218.200.188.213] login attempt [sonar/sonar] failed
2016-05-23 01:43:28+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:46109 (172.17.0.2:2222) [session:$
2016-05-23 01:43:31+0000 [SSHSservice ssh-userauth on HoneyPotTransport,4,218.200.188.213] login attempt [tuxedo/tuxedo] failed
2016-05-23 01:43:32+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:46632 (172.17.0.2:2222) [session:$
2016-05-23 01:43:35+0000 [SSHSservice ssh-userauth on HoneyPotTransport,5,218.200.188.213] login attempt [nexus/nexus] failed
2016-05-23 01:43:37+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:47171 (172.17.0.2:2222) [session:$
2016-05-23 01:43:42+0000 [SSHSservice ssh-userauth on HoneyPotTransport,6,218.200.188.213] login attempt [redmine/redmine] failed
2016-05-23 01:43:44+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:48067 (172.17.0.2:2222) [session:$
2016-05-23 01:43:46+0000 [SSHSservice ssh-userauth on HoneyPotTransport,7,218.200.188.213] login attempt [openerp/openerp] failed
2016-05-23 01:43:48+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:48538 (172.17.0.2:2222) [session:$
2016-05-23 01:43:52+0000 [SSHSservice ssh-userauth on HoneyPotTransport,8,218.200.188.213] login attempt [jenkins/jenkins] failed
2016-05-23 01:43:53+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:48786 (172.17.0.2:2222) [session:$
2016-05-23 01:43:56+0000 [SSHSservice ssh-userauth on HoneyPotTransport,9,218.200.188.213] login attempt [treino/treino] failed
2016-05-23 01:43:58+0000 [cowrie.ssh.transport.HoneyPotSSHFactory] New connection: 218.200.188.213:48939 (172.17.0.2:2222) [session:$
2016-05-23 01:44:01+0000 [SSHSservice ssh-userauth on HoneyPotTransport,10,218.200.188.213] login attempt [glassfish/glassfish] failed

```

Figure 16 - Cowrie log entries

```

GNU nano 2.2.6 File: cowrietesttest.arfff
Relation cowrietest
Attribute Port numeric
Attribute Status (succeeded,failed)
Attribute AttemptOnPort numeric
Attribute AttemptsOnIP numeric
Attribute Malicious (0,1)

#data
59753,failed,1,21,0

[ Read 10 lines (Warning: No write permission) ]

```

Figure 17 - Cowrie Machine Learning test file

The above figures show that the scripts were functioning properly as they produced the expected result.

4.4.3 Machine Learning

To show that the machine learning process was working, Figure 18 shows the file located on the Ubuntu server which contains the last classification result.

```

GNU nano 2.2.5      File: courie.txt

J48 pruned tree
-----
AttemptsOnIP <= 6: 0 (9.0)
AttemptsOnIP > 6: 1 (20.0)

Number of Leaves :    2
Size of the tree :    3

=== Error on test data ===

Correctly Classified Instances      0           0      %
Incorrectly Classified Instances    1          100      %
Kappa statistic                     0
Mean absolute error                 1
Root mean squared error             1
Total Number of Instances          1

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0.000    0.000    0.000    0.000    0.000    0.000    ?         1.000    0
      0.000    1.000    0.000    0.000    0.000    0.000    ?         ?         1
Weighted Avg.   0.000    0.000    0.000    0.000    0.000    0.000    0.000    1.000

=== Confusion Matrix ===

 a b  <-- classified as
0 1 | a = 0
0 0 | b = 1

-

Get Help      WriteOut
Exit          Justifu
Read File     Where Is
Prev Page    Next Page
Cut Text     Undo/Text
Cur Pos     To Shell

```

Figure 18 - Classification Result file

4.4.4 Web Interface

The web hosting server contains two files that are fundamental for viewing the result from a web page. These are `cowrie.txt`, shown in Figure 18, which is uploaded automatically every minute and `index.php` which contains PHP code for interpreting the result from the classification file. Figure 19 shows the interface when an attack is detected while Figure 20 shows the status for normal traffic. This was based on Cowrie honeypot traffic.

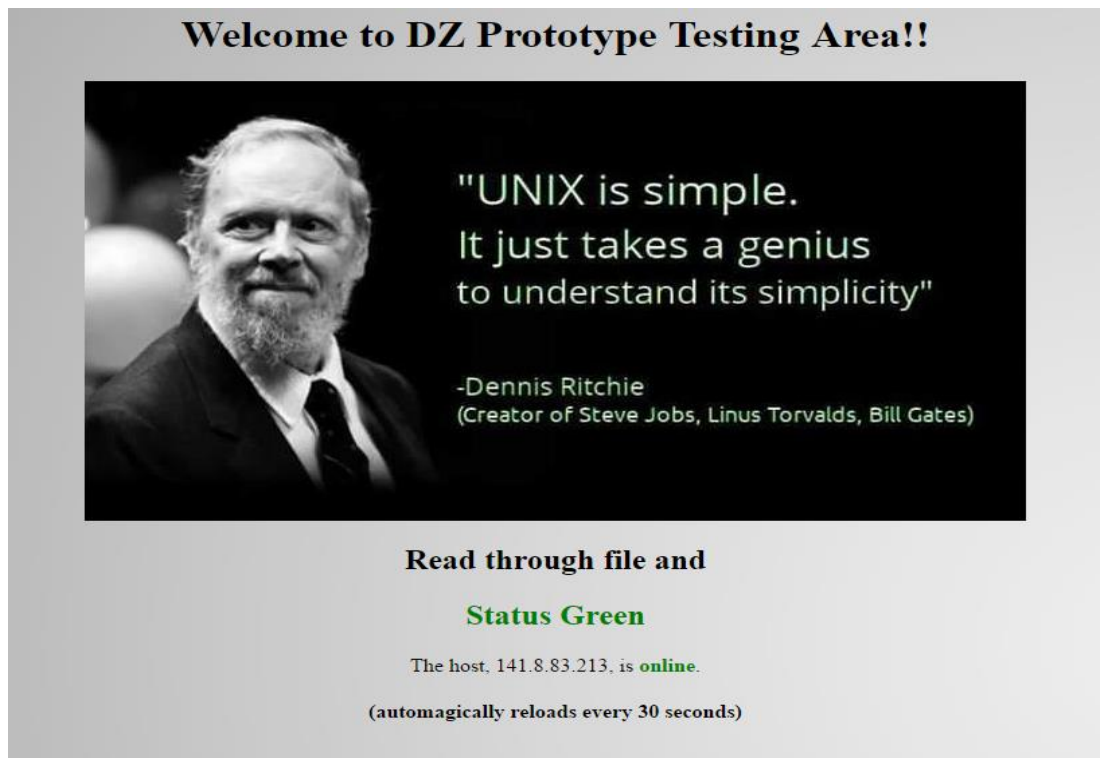


Figure 19 - Benevolent Traffic



Figure 20 - Malicious Traffic

Chapter 5. Conclusion and Future Work

5.1 Conclusion

The first objective of this dissertation was to research intrusion detection systems to identify the different techniques that were proposed for classifying traffic on a host machine or network infrastructure. The proposed work in the literature was critically analysed to highlight the advantages and disadvantages of each technique. After analysing the literature, it was identified that the next evolutionary step for intrusion detection systems rely on machine learning techniques. The second objective was to find appropriate tools for implementing an intrusion detection system that uses honeypots as data gathering mechanisms and machine learning for classification.

Subsequently, a prototype was developed reflecting the designs drawn at an earlier stage to meet the end objective of this work. A system diagram illustrates an overview of the developed prototype and the operating environment. Furthermore, UML diagrams were drawn to present all the software components. The developed prototype makes use of a particular honeypot called Cowrie, which logs login attempts and brute force attacks. Machine learning was conducted using an open source tool known as WEKA. The prototype works on traffic directed to a gateway on a home network. A web interface was also designed to filter the classification results in an understood format. Software testing was conducted to ensure that the functionality written for the prototype works as it was intended to.

This project has proved to be flexible and the system developed is effective and capable of distinguishing between malicious and non-malicious attempts. A better approach than initially proposed was established as there were sufficient mixed external attempts to test the system.

5.2 Limitations

To begin with, the main limitation of this prototype is the limited amount of attributes that are used for machine learning. Furthermore, the system was tested on a home network, limiting to a certain extent the amount of traffic targeted towards the network. Also, machine learning techniques were implemented using one tool, WEKA, and one classifier was used as time was limited. The prototype developed is near-real time, given that the automation process has a time delay of one minute.

5.3 Future Work

The prototype architecture was designed in such a way to allow extensibility to the work completed. Future work in relation to this dissertation may feature the inclusion of more honeypots for an improved intrusion detection system. An increase in attributes during feature selection would also be ideal. The use of more attributes in machine learning will increase accuracy and reliability. Moreover, it would be interesting to implement and compare different classifiers in the machine learning phase, such as support vector machine (SVM), to see which is more accurate. Finally, the proposed system can be deployed on physical hardware in a corporate network instead of a small home network.

References

- Altwaijry, H. (2011). Bayesian based intrusion detection system. *Lecture Notes in Electrical Engineering*. [Online]. 170 LNEE (1). p.pp. 29–44. Available from: <http://dx.doi.org/10.1016/j.jksuci.2011.10.001>.
- Boer, P. De & Pels, M. (2005). Host-based Intrusion Detection Systems. *Amsterdam University*.
- Bringer, M.L., Chelmecki, C.A. & Fujinoki, H. (2012). A Survey: Recent Advances and Future Trends in Honeypot Research. *International Journal of Computer Network and Information Security*. 4 (10). p.pp. 65–77.
- Cohen, F. (1998). A Note on the Role of Deception in Information Protection. *Computer & Security*. 17 (6). p.pp. 483–506.
- Denning, P.J. (1989). The ARPANET after twenty years. *American Scientist*. 77 (6). p.pp. 1–18.
- Deutsche Telekom AG (2016). *T-Pot 16.03 - Enhanced Multi-Honeypot Platform*. [Online]. 2016. Available from: <http://dtag-dev-sec.github.io/mediator/feature/2016/03/11/t-pot-16.03.html>. [Accessed: 14 May 2016].
- Elmer-DeWitt, P., Murphy, J. & Krance, M. (1983). The 414 Gang Strikes Again. *Time*. [Online]. 122 (9). p.p. 77. Available from: <http://proxyiub.uits.iu.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=54226204&site=ehost-live&scope=site>.
- Fosnock, C. (2005). Computer worms: past, present, and future. *East Carolina University*. [Online]. Available from: [http://www.hackerzvoice.net/madchat/vxdevl/avtech/Computer Worms: Past, Present, and Future.pdf](http://www.hackerzvoice.net/madchat/vxdevl/avtech/Computer%20Worms:Past,Present,andFuture.pdf).
- Gangwar, A. & Sahu, S. (2014). *OPEN ACCESS A survey on anomaly and signature based intrusion detection system (IDS)*. 4 (4). p.pp. 67–72.
- García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G. & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*. [Online]. 28 (1-2). p.pp. 18–28. Available from: <http://www.sciencedirect.com/science/article/pii/S0167404808000692>.
- Gupta, V., Singh, M. & Bhalla, V.K. (2014). *Pattern Matching Algorithms for Intrusion Detection and Prevention System : A Comparative Analysis*. p.pp. 50–54.
- Hay, A., Cid, D., Bary, R. & Northcutt, S. (2008). *OSSEC Host-Based Intrusion Detection Guide*. [Online]. Elsevier. Available from: <http://www.sciencedirect.com/science/article/pii/B9781597492409000041>. [Accessed: 9 May 2016].
- Heberlein, L., Dias, G., Levitt, K.N., Mukherjee, B., Wood, J. & Wolber, D. (1990). A network security monitor. *S&P*. [Online]. p.pp. 296 – 304. Available from:

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=63859.

- Hoque, M.S., Mukit, M.A., Bikas, M.A.N. & Sazzadul Hoque, M. (2012). An Implementation of Intrusion Detection System Using Genetic Algorithm. *International Journal of Network Security Its Applications*. [Online]. 4 (2). p.pp. 109–120. Available from: <http://www.airccse.org/journal/nsa/0312nsa08.pdf>.
- Intel (2002). *Intel® Hyper-Threading Technology*. [Online]. 2002. Available from: <http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html>. [Accessed: 14 May 2016].
- Jain, A., Sharma, S. & Sisodia, M. (2011). Network Intrusion Detection by Using Supervised and Unsupervised Machine Learning Techniques: A Survey. *International Journal of Computer ...* [Online]. 1 (3). p.pp. 14–20. Available from: http://www.ijctee.org/files/Issuethree/IJCTEE_1111_03.pdf.
- Kabila, R. (2008). *Network Based Intrusion Detection and Prevention Systems in IP-Level Security Protocols*. 2 (10). p.pp. 661–667.
- Khandagale, V. V & Kalshetty, Y. (2013). *Review and Discussion on different techniques of Anomaly Detection Based and Recent Work*. 2 (10). p.pp. 3214–3218.
- Kintana, C. (2006). *History & Impact of Hacking : Final Paper The Word ‘ Hacker ’*.
- Koch, R., Stelte, B. & Golling, M. (2012). Attack trends in present computer networks. *2012 4th International Conference on Cyber Conflict, 2012 CYCON*. p.pp. 1–12.
- Korčák, M., Lámer, J. & Jakab, F. (2014). Intrusion Prevention/Intrusion Detection System (Ips/Ids) for Wifi Networks. *International Journal of Computer Networks & Communications*. [Online]. 6 (4). p.pp. 77–89. Available from: <http://airccse.org/journal/cnc/6414cnc07.pdf>.
- Kotsiantis, S.B. (2007). Supervised machine learning: A review of classification techniques. *Informatica*. [Online]. 31. p.pp. 249–268. Available from: http://books.google.com/books?hl=en&lr=&id=vLiTXDHR_sYC&oi=fnd&pg=PA3&dq=survey+machine+learning&ots=CVsyuwYHjo&sig=A6wYWvywU8XTc7Dzp8ZdKJaW7rc\papers://5e3e5e59-48a2-47c1-b6b1-a778137d3ec1/Paper/p800\http://www.informatica.si/PDF/31-3/11_Kotsiantis-S.
- Kulhalli, K. V & Khot, S.R. (2014). *Network Based Intrusion Detection Using Honey pot Deception II System Working*. (April). p.pp. 805–810.
- Letou, K. & Devi, D. (2013). *Host-based Intrusion Detection and Prevention System (HIDPS)*. 69 (26). p.pp. 27–33.
- Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyszogrod, D., Cunningham, R.K. & Zissman, M.A. (2000). Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. *Proceedings DARPA Information Survivability Conference and Exposition*. 2. p.pp. 12–26.
- Malav, S., Avinash, M.S., Satish, N.S. & Sandeep, S.C. (2016). *Network Security Using IDS , IPS & Honeypot*. 2 (2). p.pp. 27–30.

- Marinova-Boncheva, V. (2007). A Short Survey of Intrusion Detection Systems *. *Problems of Engineering Cybernetics and Robotics*. 58. p.pp. 23–30.
- Mary, J.V. & Devi, P.G. (2013). *Snort Rule Technique For Detecting Worm Attacks*. 2 (11). p.pp. 273–277.
- Mokube, I. & Adams, M. (2007). *Honeypots : Concepts , Approaches , and Challenges*. p.pp. 321–326.
- Noel, S. & Jajodia, S. (2007). Attack Graphs for Sensor Placement , Alert Prioritization , and Attack Response. *Cyberspace Research Workshop*. p.pp. 1–8.
- Oluwatosin, H.S. (2014). Client-Server Model. *IOSR Journal of Computer Engineering*. 16 (1). p.pp. 67–71.
- Oosterhof, M. (2015). *Cowrie Honeypot*. [Online]. 2015. Available from: <http://www.micheloosterhof.com/cowrie/>. [Accessed: 23 May 2016].
- Parande, V. & Kori, P.S. (2015). Host Based Intrusion Detection System. *International Journal of Sciene and Research*. 4 (4). p.pp. 559–561.
- Portokalidis, G. & Bos, H. (2007). SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Computer Networks*. [Online]. 51 (5). p.pp. 1256–1274. Available from: <http://www.sciencedirect.com/science/article/pii/S138912860600243X>.
- Prasad.B, R., Abraham, A., Abhinav, A., Gurlahosur, S. V & Srinivasa, Y. (2011). *Design and Efficient Deployment of honeypot and dynamic rule based live network intrusion collaborative system*. 3 (2). p.pp. 52–65.
- Price, D. (2015). *Surprising Facts and Stats About The Big Data Industry »*. [Online]. 2015. Available from: <http://cloudtweaks.com/2015/03/surprising-facts-and-stats-about-the-big-data-industry/>. [Accessed: 8 May 2016].
- PwC (2015). *2015 INFORMATION SECURITY BREACHES SURVEY*.
- Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. *LISA '99: 13th Systems Administration Conference*. [Online]. p.pp. 229–238. Available from: http://static.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf.
- Scarfone, K. & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS) Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*. [Online]. 800-94. p.p. 127. Available from: <http://www.reference.com/go/http://csrc.ncsl.nist.gov/publications/nistpubs/800-94/SP800-94.pdf>.
- Sharma, H. (2013). *Crontab – Quick Reference | Admin's Choice - Choice Of Unix And Linux Administrators*. [Online]. 2013. Available from: <http://www.adminschoice.com/crontab-quick-reference>. [Accessed: 21 May 2016].
- Simon, P. (2013). *Too Big to Ignore: The Business Case for Big Data*. 1st Ed. Wiley.

- Spitzner, L. (2002). *Honeypots: Tracking Hackers*. Addison Wesley.
- Stallings, W., Bauer, M. & Hirsch, E.M. (2014). *Computer Security Principles and Practice*. 3rd Ed. United States of America: Pearson Education.
- Taylor, P. & Hayatle, O. (2013). A Markov Decision Process Model for High Interaction Honeypots. *Information Security Journal: A Global Perspective*. [Online]. 22:4, 159- (October 2014). p.pp. 37–41. Available from: <http://dx.doi.org/10.1080/19393555.2013.828802>.
- Tsai, C.F., Hsu, Y.F., Lin, C.Y. & Lin, W.Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications*. [Online]. 36 (10). p.pp. 11994–12000. Available from: <http://dx.doi.org/10.1016/j.eswa.2009.05.029>.
- Turban, E., Lee, J.K., King, D., McKay, J. & Marshall, P. (2007). Building e-commerce applications and infrastructure. In: *Electronic Commerce : A Managerial Perspective*. Pearson, p. 27.
- Uzun, E. (2014). *An Automated Bot Detection System through Honeypots for Large-Scale*. p.pp. 255–270.
- VMWARE (2014). What Is a Virtual Machine? *VMware vSphere 4 - ESX and vCenter Server*. [Online]. p.p. One. Available from: http://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.vmadmin.doc_41/vsp_vm_guide/about_vms_in_vsp_datacenter/c_what_is_a_virtual_machine.html.
- Vokorokos, L. & Baláž, A. (2010). Host-based intrusion detection system. *INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings*. p.pp. 43–47.
- Walker, B. (2015). *Every Day Big Data Statistics - 2.5 Quintillion Bytes of Data Created Daily -*. [Online]. 2015. Available from: <http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/>. [Accessed: 29 January 2016].
- Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J. & Steinberg, D. (2008). *Top Ten Algorithms in Data Mining*. 1st Ed. [Online]. Chapman and Hall/CRC. Available from: <http://link.springer.com/10.1007/s10115-007-0114-2>.
- Yadav, P. & Singh, D. (2013). *A Review on Network Intrusion Detection System*. 4 (9). p.pp. 3842–3847.
- Zou, Y. & Chakrabarty, K. (2003). Sensor Deployment and Target Localization Based on Virtual Forces. *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM)*. 2 (C). p.pp. 1293 – 1303.

Appendix A - Scripts

logretreiver.sh

```
1.  #!/bin/sh
2.
3.  echo "Retreiving all logs from their directories"
4.
5.  # Start retreiving files from their directory to put them in destination folder
6.  echo
7.  sleep 1
8.  echo "-----"
9.  #Retreiving Logs & JSON file if available
10.
11. #PATH = /home/tsec
12. #Getting Cowrie
13. sudo cp /data/cowrie/log/cowrie.log ~/alllogs
14.
15. sudo cp /data/cowrie/log/cowrie.json ~/alljson
16.
17. echo "Cowrie files successfully retreived..100%"
18.
19. sleep 1
20.
21. #Getting Dionaea
22. sudo cp /data/dionaea/log/dionaea.log ~/alllogs
23.
24. sudo cp /data/dionaea/log/dionaea.json ~/alljson
25.
26. sudo cp /data/ews/dionaea/ews.json ~/alljson
27.
28. echo "Dionaea files successfully retreived..100%"
29.
30. sleep 1
31.
32. #Getting Elasticpot
33. sudo cp /data/elasticpot/log/* ~/alllogs
34.
35.
36. echo "Elasticpot searches retreived..100%"
37.
38. sleep 1
39. #Getting glastopf
40. sudo cp /data/glastopf/log/glastopf.log ~/alllogs
41.
42. sleep 1
```

```

43.
44. echo "Glastopf files retrieved..100%"
45. #Getting HoneyTrap (check attacks folder and copy all registered attacks maybe?)
46. sudo cp /data/honeytrap/log/attacker.log ~/alllogs
47.
48. sudo cp /data/honeytrap/log/honeytrap.log ~/alllogs
49.
50. sleep 1
51.
52. echo "HoneyTrap files retrieved..100%"
53. #Getting Suricata
54. sudo cp /data/suricata/log/eve.json ~/alljson
55. sudo cp /data/suricata/log/p0f.json ~/alljson
56.
57. echo "Suricata files retrieved..100%"
58.
59. #Open connection with Honeydrive & enter into shell
60.
61. #ssh honeydrive@192.168.0.18
62.
63. #Syncing and transferring files to a remote linux machine a.k.a HoneyDrive III
64.
65. #sudo rsync -a --progress ~/alllogs/*
   honeydrive@192.168.0.18:~/Desktop/Transfer/Logs
66. #sudo rsync -a --progress ~/alljson/*
   honeydrive@192.168.0.18:~/Desktop/Transfer/JSON
67. echo
68.
69. #copying files from root directory to tsec directory
70. #if run manually from tsec user gives same file warning
71. sudo cp ~/alljson/* /home/tsec/alljson
72. sudo cp ~/alllogs/* /home/tsec/alllogs
73.
74. #this is being done in extractor script
75. #sudo cp ~/alllogs/* /home/tsec/prototype/logs
76.
77. echo "Files have been transferred to the secret location"
78.
79. echo
80.
81. echo "Script run successfully :)"
82.
83. echo "-----"
84.
85. #clearing previous text for easier maintainability
86. > /home/tsec/prototype/debuglogs/retrieveput.log
87.
88.
89. echo "Retrieve successful: $(date)" >>
   /home/tsec/prototype/debuglogs/retrieveput.log

```

```

90.
91. echo $(date)
92. echo
93. # need for rsync or else copy will do just fine to overwrite the data?
94. #### use rsync better
95.
96. #we can rsync to a remote host once we cron this script. Check if copy replaces
97.
98. # From this point forward we use ftp to send all these files to somewhere outside the machine

```

extractor.sh

```

1. #!/bin/bash
2.
3. #copies logs from tsec/root to prototype/logs
4. #if runs in root, no prob
5. sudo cp ~/alllogs/* /home/tsec/prototype/logs
6.
7. #extracts the last ten entries from file
8.
9. #sudo grep "GMT" /home/tsec/prototype/logs/attacker.log | tail -n 10 >  

/home/tsec/prototype/logs/extractedlogs/attackerresult.log
10. #sudo grep "GMT" /home/tsec/prototype/logs/attacker.log | uniq | tail -n 10 >  

/home/tsec/prototype/logs/extractedlogs/attackerresult.log
11.
12.
13. #we can add a second match phrase to this if it needs be word1|word2
14. #apparently grep is case sensitive
15. #sudo grep "New connection" /home/tsec/prototype/logs/cowrie.log | grep "Remote  

SSH version" | tail -n 10 > /home/tsec/prototype/logs/extractedlogs/cowrieresult.log
16. #maybe we can add second search string for getting remote ssh version?
17.
18. sudo grep 'New connection|login attempt' /home/tsec/prototype/logs/cowrie.log |  

uniq -u | tail -n 100 > /home/tsec/prototype/logs/extractedlogs/cowrieresult.log
19.
20.
21.
22. #sudo grep "alert" /home/tsec/prototype/logs/elasticpot.log | sort -u -t, -k2,8 | tail -n  

10 > /home/tsec/prototype/logs/extractedlogs/elasticpotresult.log
23.
24. #this glastopf grep was chosen since we need an IP to be classified as non-  

malicious.emy ip
25. #sudo grep 'GET|POST' /home/tsec/prototype/logs/glastopf.log | sort -uk4,4 | tac |  

sort -k4,4 | sort -k1,2 | tail -n 20 >  

/home/tsec/prototype/logs/extractedlogs/glastopfresult.log

```

```

26.
27.
28. sudo grep 'GET|POST' /home/tsec/prototype/logs/glastopf.log | sort -uk7 | tail -n
    20 > /home/tsec/prototype/logs/extractedlogs/glastopfresult.log
29.
30.
31. #sudo grep "GET" /home/tsec/prototype/logs/glastopf.log | grep -vF 141.8.83.213 |
    sort -k4,4 | tac | sort -uk4,4 | sort -k1,2 | tail -n 10 >
    /home/tsec/prototype/logs/extractedlogs/glastopfresult.log
32.
33.
34. #clearing previous text for easier readability
35. > /home/tsec/prototype/debuglogs/extractorput.log
36.
37. echo "Filter Operations Complete: $(date)" >>
    /home/tsec/prototype/debuglogs/extractorput.log 2>&1
38.
39.
40. echo "Terminal Out OK!"

```

prototype.prg

```

1. #!/usr/bin/perl -w
2.
3. $path1 = "/home/tsec/prototype/logs/extractedlogs/attackerresult.log";
4. $path2 = "/home/tsec/prototype/logs/extractedlogs/cowrieresult.log";
5. $path3 = "/home/tsec/prototype/logs/extractedlogs/elasticpotresult.log";
6. $path4 = "/home/tsec/prototype/logs/extractedlogs/glastopfresult.log";
7. $honeytrap = ">/home/tsec/prototype/honeycsv/honeytrap.csv";
8. $cowrie = ">/home/tsec/prototype/honeycsv/cowrie.csv";
9. $elasticpot = ">/home/tsec/prototype/honeycsv/elasticpot.csv";
10. $glastopf = ">/home/tsec/prototype/honeycsv/glastopf.csv";
11.
12. $end = "";
13.
14. #function definition #Pattern for attackerlog only
15. sub honeytrapExtractor(){
16. if(open(FILE1, $honeytrap) or die "Can't open '$honeytrap': $!"){
17.
18. #
19.
20. }
21. else{ #the code over here is useless given that the ">dir" create file even if doesnt
    exists.
22. my $existingDirectory = "~/prototype";

```

```

23. mkdir $existingDirectory unless -d $existingDirectory; #checking if dir already exists,
    else make it
24.
25. open(FILE2, ">", "$existingDirectory/attacker.csv");
26. open(LOG, $path1) or die "Can't open 'path1': $!";
27.
28. print FILE2 "Protocol,SourceIP,SourcePort,FileSize,Malicious\n";
29.
30. while(my $lines = <LOG>){
31.
32. my ($protocol, $ip, $port, $size) = (split /\s:()/+/, $lines)[6,7,8,-2];
33.
34. print FILE2 join " ", ($protocol, $ip, $port, $size, $end);
35. print FILE2 "\n";
36. }
37.
38. }
39.
40. #open(FILE, $attacker) or die "Can't open '$attacker': $!";
41.
42. print FILE1 "Protocol,SourcePort,FileSize,Malicious\n";
43. open(LOG, $path1) or die "Can't open '$path1': $!";
44. while(my $lines = <LOG>){
45.
46. #my ($protocol, $ip, $port, $size) = (split /\s:()/+/, $lines)[6,7,8,-2];
47. my($protocol, $port, $size) = (split /\s:()/+/, $lines)[6,8,-2];
48. print FILE1 join " ", ($protocol, $port, $size, $end);
49. print FILE1 "\n";
50.
51.
52. }
53. }
54.
55.
56. sub cowrieExtractor(){
57.
58. open(FILE2, $cowrie) or die "Can't open '$cowrie': $!";
59.
60. open(LOG2, $path2) or die "Can't open '$path2': $!";
61.
62. my (%rept, %ip_tot);
63. my ($ip, $port);
64.
65. while (my $line = <LOG2>)
66. {
67. if ($line =~ /New connection/) {
68. ($ip, $port) = $line =~ /New connection:\s+([^\s:]+):\s+(\d+)/;
69. next;
70. }
71. elsif (!$ip or !$port) { next } # First lines come before New connection

```



```

72.
73. my ($usr, $status) = $line =~ m/login attempt\s+\[([^\]]+)\]\s+(\lw+)/;
74. if ($usr and $status) {
75.     $rept{$port}{$ip}{$usr}{$status}++;
76.     $ip_tot{$ip}{$status}++;
77. }
78. else { warn "Line with an unexpected format:\n$line" }
79. }
80.
81. print FILE2 "Port,Status,AttemptOnPort,AttemptsOnIP,Malicious\n";
82. foreach my $port (sort keys %rept) {
83.     foreach my $ip (sort keys %{$rept{$port}}) {
84.         foreach my $usr (sort keys %{$rept{$port}{$ip}}) {
85.             foreach my $stat ( sort keys %{$rept{$port}{$ip}{$usr}} ) {
86.                 print FILE2 "$port,$stat,$rept{$port}{$ip}{$usr}{$stat}";
87.                 print FILE2 ",$ip_tot{$ip}{$stat},\n";
88.             }
89.         }
90.     }
91. }
92.
93. #prints IP and Number of Occurrences based on that IP for testing purposes
94.
95. #print "\n";
96. #print "IP,Status,Occurences\n";
97. #foreach my $ip (sort keys %ip_tot) {
98. # foreach my $stat ( sort keys %{$ip_tot{$ip}} ) {
99. # print "$ip,$stat,$ip_tot{$ip}{$stat}\n";
100.     # }
101.     #}
102.
103.
104.     # if all variables are not equal to null, then print to file
105.     #if($ip && $port && $usr && $pass && $status ne ""){
106.     #print FILE2 join " ",($ip, $port, $usr, $pass, $status);
107.     #print FILE2 "\n";
108.
109.
110.
111.     }
112.
113.     sub counter(){
114.
115.         $result = 0;
116.         #open(FILE2, $cowrie) or die "Can't open '$cowrie': $!";
117.         while(my $otherlines = <LOG2>){
118.
119.             if($otherlines =~ /login attempt/){
120.                 ($user, $password) = (split /\s:\/\]/+/ , $otherlines)[-3,-2];
121.                 if($_[1] =~ /$user/ && $_[2] =~ /$password/){

```

```

122.     $result++;
123. }#if ip matches i think i have to do this with split
124.
125.     #print "TEST\n";
126. }
127.     #print "Combo $_[0] and $_[1]\n";
128.
129. }
130.     #print "$result";
131.     return $result;
132. }
133.
134.
135.     sub elasticpotExtractor(){
136.
137.         open(FILE3, $elasticpot) or die "Can't open '$elasticpot': $!";
138.         open(LOG3, $path3) or die "Can't open 'path3': $!";
139.
140.         #attributes here
141.         print FILE3
142.         "EventType,SourcePort,DestinationPort,HoneypotName,Malicious\n";
143.
144.         while(my $lines = <LOG3>){
145.             my($type, $ip, $port, $destip, $destport, $potname) = (split /[{}":,]+/,
146.             $lines)[6,8,10,12,14,17];
147.
148.             print FILE3 join " ",($type, $port, $destport, $potname, $end);
149.             print FILE3 "\n";
150.         }
151.
152.     }
153.
154.     sub glastopfExtractor(){
155.
156.         open(FILE4, $glastopf) or die "Can't open '$glastopf': $!";
157.         open(LOG4, $path4) or die "Can't open '$path4': $!";
158.
159.         #attributes here(get srcip for now and what was actually posted or
160.         requested[get])
161.         #it does not matter if we get the same ip no of time as long as the get is
162.         unique
163.         print FILE4 "Method,ContentRequested,Malicious\n";
164.
165.         while(my $lines = <LOG4>){
166.
167.             my($ip, $method, $content, $target) = (split /[s:]+/, $lines)[6,7,8,10];
168.
169.             print FILE4 join " ",($method, $content, $end);

```

```

168.     print FILE4 "\n";
169.
170. }
171.
172. }
173.
174.     close(LOG);
175.     close(FILE1);
176.     close(FILE2);
177.
178.     close(LOG2);
179.     close(FILE2);
180.
181.     close(LOG3);
182.     close(FILE3);
183.
184.     close(LOG4);
185.     close(FILE4);
186.
187.     #honeytrapExtractor();
188.     cowrieExtractor();
189.     #elasticpotExtractor();
190.     #glastopfExtractor();
191.     print "Feature Selection and formatting complete!!\n";

```

converter.sh

```

1.  #NOTE: We might have to give the exact path since we are running from root.
2.
3.
4.  #These are for training #WE RUN THESE ONCE. THEN ALWAYS APPEND
5.
6.  #Converter cmd for Cowrie
7.  #var=$(echo -e "\052") #only used if we get raw usr and password
8.  path=/home/tsec
9.
10. #RUN ONLY ONCE THEN MODIFY MALICIOUS TO {0,1}
11. #sudo java -cp $path/prototype/weka-3-9-0/weka.jar
    weka.core.converters.CSVLoader $path/prototype/honeycsv/cowrie.csv >
    $path/prototype/honeycsv/trainfiles/cowrie.arff
12.
13. #Converter for Glastopf
14.
15. #sudo java -cp $path/prototype/weka-3-9-0/weka.jar
    weka.core.converters.CSVLoader $path/prototype/honeycsv/glastopf.csv >
    $path/prototype/honeycsv/trainfiles/glastopf.arff

```

```

16.
17. #Converter for Elasticpot
18.
19. #sudo java -cp $path/prototype/weka-3-9-0/weka.jar
    weka.core.converters.CSVLoader $path/prototype/honeycsv/elasticpot.csv >
    $path/prototype/honeycsv/trainfiles/elasticpot.arff
20.
21. #Converter for Honeytrap
22.
23. #sudo java -cp $path/prototype/weka-3-9-0/weka.jar
    weka.core.converters.CSVLoader $path/prototype/honeycsv/honeytrap.csv >
    $path/prototype/honeycsv/trainfiles/honeytrap.arff
24. echo test
25.
26.
27. #Now we call the J48 classifier to work on training files to create models #WE DO
    THIS ONLY ONCE UNLESS NEEDED TO ADD A NEW ENTRY
28. #if this gives error remove cp
29. sudo java -cp $path/prototype/weka-3-9-0/weka.jar weka.classifiers.trees.J48 -t
    $path/prototype/honeycsv/trainfiles/cowrie.arff -d
    $path/prototype/honeycsv/models/cowrie.model
30.
31. #java -cp $path/prototype/weka-3-9-0/weka.jar weka.classifiers.trees.J48 -t
    $path/prototype/honeycsv/trainfiles/glastopf.arff -d
    $path/prototype/honeycsv/models/glastopf.model
32.
33. #java -cp $path/prototype/weka-3-9-0/weka.jar weka.classifiers.trees.J48 -t
    $path/prototype/honeycsv/trainfiles/elasticpot.arff -d
    $path/prototype/honeycsv/models/elasticpot.model
34.
35. #java -cp $path/prototype/weka-3-9-0/weka.jar weka.classifiers.trees.J48 -t
    $path/prototype/honeycsv/trainfiles/honeytrap.arff -d
    $path/prototype/honeycsv/models/honeytrap.model
36. echo test2
37.
38. echo "Operation Complete $date" > $path/prototype/debuglogs/converterput.log

```

extractortest.sh

```

1. #!/bin/bash
2.
3. #copies logs from tsec/root to prototype/logs
4. #if runs in root, no prob
5. sudo cp ~/alllogs/* /home/tsec/prototype/logs
6.
7. #extracts the last ten entries from file

```

- 8.
9. `#sudo grep "GMT" /home/tsec/prototype/logs/attacker.log | tail -n 10 > /home/tsec/prototype/logs/extractedlogs/attackerresult.log`
10. `#sudo grep "GMT" /home/tsec/prototype/logs/attacker.log | uniq | tail -n 1 > /home/tsec/prototype/logs/extractedlogs/attackerresulttest.log`
- 11.
- 12.
13. *#we can add a second match phrase to this if it needs be word1|word2*
14. *#apparently grep is case sensitive*
15. `#sudo grep "New connection" /home/tsec/prototype/logs/cowrie.log | grep "Remote SSH version" | tail -n 10 > /home/tsec/prototype/logs/extractedlogs/cowrieresult.log`
16. *#maybe we can add second search string for getting remote ssh version?*
- 17.
18. `sudo grep 'New connection|login attempt' /home/tsec/prototype/logs/cowrie.log | uniq -u | tail -n 100 > /home/tsec/prototype/logs/extractedlogs/cowrieresulttest.log`
- 19.
20. `sudo cp /home/tsec/prototype/honeycsv/testfiles/cowrietest.arff /home/tsec/prototype/honeycsv/testfiles/cowrietesttest.arff`
- 21.
22. `#sudo grep "alert" /home/tsec/prototype/logs/elasticpot.log | sort -u -t, -k2,8 | tail -n 1 > /home/tsec/prototype/logs/extractedlogs/elasticpotresulttest.log`
- 23.
24. *#this glastopf grep was chosen since we need an IP to be classified as non-malicious.emy ip*
25. `#sudo grep 'GET|POST' /home/tsec/prototype/logs/glastopf.log | sort -k4,4 | tac | sort -uk4,4 | sort -k1,2 | tail -n 1 > /home/tsec/prototype/logs/extractedlogs/glastopfresulttest.log`
- 26.
27. `#sudo grep "GET" /home/tsec/prototype/logs/glastopf.log | grep -vF 141.8.83.213 | sort -k4,4 | tac | sort -uk4,4 | sort -k1,2 | tail -n 10 > /home/tsec/prototype/logs/extractedlogs/glastopfresult.log`
- 28.
- 29.
30. *#clearing previous text for easier readability*
31. `> /home/tsec/prototype/debuglogs/extractorputtest.log`
- 32.
33. `echo "Filter Operations Complete: $(date)" >> /home/tsec/prototype/debuglogs/extractorputtest.log 2>&1`
- 34.
- 35.
36. `echo "Terminal Out OK!"`

prototypetest.prg

1. `#!/usr/bin/perl -w`
- 2.
3. `$path1 = "/home/tsec/prototype/logs/extractedlogs/attackerresulttest.log";`
4. `$path2 = "/home/tsec/prototype/logs/extractedlogs/cowrieresulttest.log";`

```

5. $path3 = "/home/tsec/prototype/logs/extractedlogs/elasticpotresulttest.log";
6. $path4 = "/home/tsec/prototype/logs/extractedlogs/glastopfresulttest.log";
7. $honeytrap = ">/home/tsec/prototype/honeycsv/honeytraptest.csv";
8. $testcowrie = "+>/home/tsec/prototype/honeycsv/testcowrie.log";
9. $cowrie = ">>/home/tsec/prototype/honeycsv/testfiles/cowrietesttest.arff";
10. $elasticpot = ">/home/tsec/prototype/honeycsv/elasticpottest.csv";
11. $glastopf = ">/home/tsec/prototype/honeycsv/glastopftest.csv";
12.
13.
14.
15. $end = "0";
16.
17. #function definition #Pattern for attackerlog only
18. sub honeytrapExtractor(){
19. if(open(FILE1, $honeytrap) or die "Can't open '$honeytrap': $!");
20.
21. #
22.
23. }
24. else{ #the code over here is useless given that the ">dir" create file even if doesnt exists.
25. my $existingDirectory = "~/prototype";
26. mkdir $existingDirectory unless -d $existingDirectory; #checking if dir already exists, else make it
27.
28. open(FILE2, ">", "$existingDirectory/honeytraptest.csv");
29. open(LOG, $path1) or die "Can't open 'path1': $!";
30.
31. print FILE2 "Protocol,SourceIP,SourcePort,FileSize,Malicious\n";
32.
33. while(my $lines = <LOG>){
34.
35. my ($protocol, $ip, $port, $size) = (split /\s:()/+/, $lines)[6,7,8,-2];
36.
37. print FILE2 join " ", ($protocol, $ip, $port, $size, $end);
38. print FILE2 "\n";
39. }
40.
41. }
42.
43. #open(FILE, $attacker) or die "Can't open '$attacker': $!";
44.
45. print FILE1 "Protocol,SourcePort,FileSize,Malicious\n";
46. open(LOG, $path1) or die "Can't open '$path1': $!";
47. while(my $lines = <LOG>){
48.
49. my ($protocol, $port, $size) = (split /\s:()/+/, $lines)[6,8,-2];
50.
51. print FILE1 join " ", ($protocol, $port, $size, $end);
52. print FILE1 "\n";

```

```

53.
54.
55. }
56. }
57.
58.
59. sub cowrieExtractor(){
60.
61. open(FILE2, $testcowrie) or die "Can't open '$testcowrie': $!";
62.
63. open(LOG2, $path2) or die "Can't open '$path2': $!";
64.
65.
66. my (%rept, %ip_tot);
67. my ($ip, $port);
68.
69. while (my $line = <LOG2>)
70. {
71. if ($line =~ /New connection/) {
72. ($ip, $port) = $line =~ /New connection:\s+([^\s]+):\s+(\d+)/;
73. next;
74. }
75. elsif (!$ip or !$port) { next } # First lines come before New connection
76.
77. my ($usr, $status) = $line =~ m/login attempt\s+V([^\s]+)\s+(\w+)/;
78. if ($usr and $status) {
79. $rept{$port}{$ip}{$usr}{$status}++;
80. $ip_tot{$ip}{$status}++;
81. }
82. else { warn "Line with an unexpected format:\n$line" }
83. }
84.
85.
86. #print FILE2 "Port,Status,AttemptOnPort,AttemptsOnIP,Malicious\n";
87. foreach my $port (sort keys %rept) {
88. foreach my $ip (sort keys %{$rept{$port}}) {
89. foreach my $usr (sort keys %{$rept{$port}{$ip}}) {
90. foreach my $stat ( sort keys %{$rept{$port}{$ip}{$usr}} ) {
91. print FILE2 "$port,$stat,$rept{$port}{$ip}{$usr}{$stat},";
92. print FILE2 "$ip_tot{$ip}{$stat},$end\n";
93. }
94. }
95. }
96. }
97. #close (FILE2);
98.
99. #open (FILE2, $testcowrie);
100.      seek FILE2, 0, 0;
101.      chomp(my @lines = <FILE2>);
102.      my $last_one = pop @lines;

```

```

103.
104.     open (FILE10, $cowrie) or die "Can't open '$cowrie': $!";
105.
106.     print FILE10 "$last_one\n";
107.     close (FILE10);
108.
109.
110.     #print "\n";
111.     #print "IP,Status,Occurences\n";
112.     #foreach my $ip (sort keys %ip_tot) {
113.     # foreach my $stat ( sort keys %{$ip_tot{$ip}} ) {
114.     # print "$ip,$stat,$ip_tot{$ip}{$stat}\n";
115.     # }
116.     #}
117.
118.
119.
120.     # if all variables are not equal to null, then print to file
121.     #if($ip && $port && $usr && $pass && $status ne ""){
122.     #print FILE2 join " ",($ip, $port, $usr, $pass, $status);
123.     #print FILE2 "\n";
124.
125.     }
126.
127.     sub elasticpotExtractor(){
128.
129.     open(FILE3, $elasticpot) or die "Can't open '$elasticpot': $!";
130.     open(LOG3, $path3) or die "Can't open 'path3': $!";
131.
132.     #attributes here
133.     print FILE3
"EventType,SourcePort,DestinationPort,HoneypotName,Malicious\n";
134.
135.     while(my $lines = <LOG3>){
136.
137.     my($type, $ip, $port, $destip, $destport, $potname) = (split /[{}":,]+/,
$lines)[6,8,10,12,14,17];
138.
139.     print FILE3 join " ",($type, $port, $destport, $potname, $end);
140.     print FILE3 "\n";
141.
142.     }
143.
144.     }
145.
146.     sub glastopfExtractor(){
147.
148.     open(FILE4, $glastopf) or die "Can't open '$glastopf': $!";
149.     open(LOG4, $path4) or die "Can't open '$path4': $!";
150.

```



```

151.      #attributes here(get srcip for now and what was actually posted or
      requested[get])
152.      #it does not matter if we get the same ip no of time as long as the get is
      unique
153.      print FILE4 "Method,ContentRequested,Target,Malicious\n";
154.
155.      while(my $lines = <LOG4>){
156.
157.          my($ip, $method, $content, $target) = (split /\s+/, $lines)[3,5,6,-2];
158.
159.          print FILE4 join " ",($method, $content, $target, $send);
160.          print FILE4 "\n";
161.
162.      }
163.
164.      }
165.
166.      $logoutput = ">/home/tsec/prototype/debuglogs/prototypetestput.log";
167.
168.      open(FILE5, $logoutput) or die "Can't open '$logoutput' :$!";
169.      $date = localtime();
170.      print FILE5 "Operation successful @ $date";
171.      close(FILE5);
172.
173.      close(LOG);
174.      close(FILE1);
175.      close(FILE2);
176.
177.      close(LOG2);
178.      close(FILE2);
179.
180.      close(LOG3);
181.      close(FILE3);
182.
183.      close(LOG4);
184.      close(FILE4);
185.
186.      honeytrapExtractor();
187.      cowrieExtractor();
188.      elasticpotExtractor();
189.      glastopfExtractor();

```

convertertest.sh

```

1.  #!/bin/bash
2.
3.  #NOTE: We might have to give the exact path since we are running from root. Hope
      not

```

- 4.
- 5.
6. `path=/home/tsec`
7. `date=$(date)`
- 8.
9. *#Now that all have been converted, next step is to train using J48 Classifier BUT to do this we need to do what we have done for now for test data*
- 10.
11. `#sudo java -cp $path/prototype/weka-3-9-0/weka.jar
weka.core.converters.CSVLoader $path/prototype/honeycsv/cowrietest.csv -F "$var"
> $path/prototype/honeycsv/testfiles/cowrietest.arff`
- 12.
13. `#sudo java -cp $path/prototype/weka-3-9-0/weka.jar
weka.core.converters.CSVLoader $path/prototype/honeycsv/elasticpottest.csv >
$path/prototype/honeycsv/testfiles/elasticpottest.arff`
- 14.
15. `#sudo java -cp $path/prototype/weka-3-9-0/weka.jar
weka.core.converters.CSVLoader $path/prototype/honeycsv/glastopf.test.csv >
$path/prototype/honeycsv/testfiles/glastopf.test.arff`
- 16.
17. `#sudo java -cp $path/prototype/weka-3-9-0/weka.jar
weka.core.converters.CSVLoader $path/prototype/honeycsv/honeytrap.test.csv >
$path/prototype/honeycsv/testfiles/honeytrap.test.arff`
- 18.
- 19.
20. *#Use model to put it against testing files #dru has to tell me how to get the output*
21. `#java -cp weka.classifiers.trees.J48 -I
$path/prototype/honeycsv/models/honeytrap.model -T
$path/prototype/honeycsv/testfiles/honeytrap.test.arff >
$path/prototype/honeycsv/results/honeytrap.txt`
- 22.
23. `#java -cp weka.classifiers.trees.J48 -I
$path/prototype/honeycsv/models/elasticpot.model -T
$path/prototype/honeycsv/testfiles/elasticpottest.arff >
$path/prototype/honeycsv/results/elasticpot.txt`
- 24.
25. `#java -cp weka.classifiers.trees.J48 -I
$path/prototype/honeycsv/models/glastopf.model -T
$path/prototype/honeycsv/testfiles/glastopf.test.arff >
$path/prototype/honeycsv/results/glastopf.txt`
- 26.
27. `java -cp $path/prototype/weka-3-9-0/weka.jar weka.classifiers.trees.J48 -I
$path/prototype/honeycsv/models/cowrie.model -T
$path/prototype/honeycsv/testfiles/cowrietesttest.arff >
$path/prototype/honeycsv/results/cowrie.txt`
- 28.
29. `sudo cp $path/prototype/honeycsv/results/cowrie.txt /root/`
- 30.
31. `#sudo rm $path/prototype/honeycsv/testfiles/cowrietesttest.arff`
- 32.

```

33.
34.
35.
36.
37. #java -cp weka.classifiers.trees.J48 -t
    $path/prototype/honeycsv/trainfiles/honeytrap.arff -T
    $path/prototype/honeycsv/testfiles/honeytraptest.arff > $path/plswork.log 2>&1
38.
39.
40. echo "Operation Complete $date" >>
    $path/prototype/debuglogs/converterputtest.log

```

uploader.sh

```

1. #!/bin/bash
2.
3.
4. HOST='ftp.byethost31.com'
5. USER='b31_17942337'
6. PASSWD='zammit20'
7.
8. ftp -n $HOST <<END_SCRIPT
9. quote USER $USER
10. quote PASS $PASSWD
11. passive
12. cd /htdocs
13. send cowrie.txt
14. quit
15. END_SCRIPT
16. exit 0

```

Appendix B - Web Interface

index.php

```
1. <html>
2. <head>
3. <title>DZ Prototype</title>
4. <link rel="icon" type="img/ico" href="images/favicon.jpg">
5. <meta http-equiv="refresh" content="30">
6. <style>
7. body {
8. background-image:url("https://www.greenville.edu/digital-signage/default-
   backgrounds/greybg1.png");
9. }
10. a:link {
11. color: white;
12. text-decoration: none;
13. }
14.
15. /* visited link */
16. a:visited {
17. color: white;
18. text-decoration: none;
19. }
20.
21. /* mouse over link */
22. a:hover {
23. color: black;
24. text-decoration: none;
25. }
26.
27. /* selected link */
28. a:active {
29. color: yellow;
30. text-decoration: none;
31. }
32.
33. footer
34. {
35.
36. }
37.
38. footer *
39. {
40. display: block;
41. }
42.
43. #footer {
44. background:#ffab62;
```

```

45. width:100%;
46. height:100px;
47. position:absolute;
48. bottom:0;
49. left:0;
50. }
51. h3 {
52. text-align:center;
53. }
54. </style>
55. </head>
56. <body>
57. <center>
58. <h1>Welcome to DZ Prototype Testing Area!!</h1>
59. </center>
60. <p></p>
61. <p></p>
62. <p></p>
63. <p style="text-align:center"></p>
64. <p></p>
65. <p></p>
66.
67. <div align="center">
68. <?php
69.
70. $filename = "cowrie.txt";
71. $line = file($filename);
72.
73.
74. if(file_exists($filename)){
75. echo "<h2>Read through file and</h2>";
76. }
77. else{
78. echo "<h2>Upload not successful</h2>";
79. }
80.
81.
82.
83. if(trim($line[15]) == "Correctly Classified Instances 0 0 %") {
84. echo "<h2><font color='red'>Last entry is a Possible Malicious Login
    Attempt</font></h2>";
85. } else {
86. echo "<h2><font color='green'>Status Green</font></h2>";
87. }
88.
89. function pingAddress($ip) {
90. $pingresult = exec("/bin/ping -n 3 $ip", $outcome, $status);
91. if (0 == $status) {

```

```

92. $status = "<font color='green'><b>online</b></font>.";
93. } else {
94. $status = "<font color='red'><b>offline</b></font>.";
95. }
96. echo "The host, $ip, is ".$status;
97. }
98.
99. pingAddress("141.8.83.213")
100.     ?>
101.     <h4><font color='black'>(automagically reloads every 30 seconds)</h4>
102.     </div>
103.     <div id="footer">
104.     <footer>
105.     <h3>Created by: Daniel Zammit</h3>
106.     <h3><a
href="https://mail.google.com/mail/?view=cm&fs=1&to=dzamm20@gmail.com&su=Hello%20Daniel!" target="_blank">Contact</a></h3>
107.     </footer>
108.     </div>
109.     </body>
110.     </html>

```