Framework for Evaluation of Network-Based Intrusion Detection System

Owen Lo

Submitted in partial fulfilment of
the requirements of Napier University for the Degree of
Bachelor of Engineering with Honours in Computer
Networks and Distributed Systems

November 2009

Supervisor: Prof William Buchanan

Second Mark: Dr Jamie Graves

Authorship Declaration

I, Owen Lo, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date:

Matriculation no: 05002961

Acknowledgements

I would like to thank Prof Bill Buchanan for giving me the opportunity to carry out this project along with all the help he has provided me.

I would like to thank Jamie Graves for being my second marker. Additionally, I would like to thank Lionel Saliou who provided feedback and help during the initial stages of this project.

Finally, I would like to thank my family for their support.

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Contents

List of Tables

List of Figures

Abstract

There are a multitude of threats now faced in computer networks such as viruses, worms, trojans, attempted user privilege gain, data stealing and denial of service. As a first line of defence, firewalls can be used to prevent threats from breaching a network. Although effective, threats can inevitably find loopholes to overcome a firewall. As a second line of defence, security systems, such as malicious software scanners may be put in place to detect a threat inside the network. However, such security systems cannot necessary detect all known threats, therefore a last line of defence comes in the form of logging a threat using Intrusion Detection Systems (Buchanan, 2009, p. 43).

Being the last line of defence, it is vital that IDSs are up to an efficient standard in detecting threats. Researchers have proposed methodologies for the evaluation of IDSs but, currently, no widely agreed upon standard exists (Mell, Hu, Lippmann, Haines, & Zissman, 2003, p. 1). Many different categories of IDSs are available, including host-based IDS (HIDS), network-based IDS (NIDS) and distributed-based IDS (DIDS). Attempting to evaluate these different categories of IDSs using a standard accepted methodology allows for accurate benchmarking of results. This thesis reviews four existing methodologies and concludes that the most important aspects in an effective evaluation of IDSs must include realistic attack and background traffic, ease of automation and meaningful metrics of evaluation.

A prototype framework is proposed which is capable of generating realistic attacks including surveillance/probing, user privilege gain, malicious software and denial of service. The framework also has the capability of background traffic generation using static network data sets. The detection metrics of efficiency, effectiveness and packet loss are defined along with resource utilisation metrics in the form of CPU utilisation and memory usage. A GUI developed in Microsoft .NET C# achieves automation of sending attack and background traffic, along with the generation of detection metrics from the data logged by the IDS under evaluation.

Using a virtual networking environment, the framework is evaluated against the NIDS Snort to show the capabilities of the implementation. Mono was used to run the .NET application in a Linux environment. The results showed that, whilst the NIDS is highly effective in the detection of attacks (true-positives), its main weakness is the dropping of network packets at higher CPU utilisations due to high traffic volume. At around 80Mbps playback volumes of background traffic and above, it was found that Snort would begin to drop packets. Furthermore, it was also found that the NIDS is not very efficient as it tends to raise a lot of alerts even when there are no attacks (false-positives).

The conclusion drawn in this thesis is that the framework is capable of carrying out an evaluation of an NIDS. However, several limitations to the current framework are also identified. One of the key limitations is that there is a need for controlled aggregation of network traffic in this framework so that attack and background traffic can be more realistically mixed together. Furthermore, the thesis shows that more research is required in the area of background traffic generation. Although the framework is capable of generating traffic using state data sets, a more ideal solution would be an implementation in which allows the user to select certain “profiles” of network traffic. This would serve the purpose of better reflecting the network environment in which the IDS will be deployed on.

1 Introduction

1.1 Project Overview

It is stated in the work of Mell, Hu, Lippmann, Haines, & Zissman (2003) that "while intrusion detection systems are becoming ubiquitous defences in today's networks, currently we have no comprehensive and scientifically rigours methodologies to test the effectiveness of these systems (Mell, *et al.*, 2003, p. 1)." In other words, a standard accepted methodology is still a requirement in the evaluation of IDSs.

The aim of this project is to produce a prototype framework which is capable of carrying out an evaluation of an NIDS. The design of the framework is achieved by attempting to apply the strengths of existing methodologies proposed by researchers whilst avoiding the limitations as best as possible. The use of meaningful metrics of evaluation for this framework are also been taken into consideration. The implementation is carried out using Microsoft .NET C# and ran under a Linux environment. The framework is then used to evaluate the NIDS Snort in order to establish the effectiveness of the approach taken.

1.2 Background

Burglar alarms, smoke alarms, fire alarms and closed circuit television all fall under the category of real life, physical intrusion detection systems (Del Carlo, Lakes, Illinois, & Practical, 2003, p.4). The purpose of these devices is to monitor specific threats, and, if the threat occurs, then either produce some form of alert (such as emitting a high pitched noise in the case of fire and burglar alarms) or log the activity (in the case of closed circuit television cameras). Examples of threats include fires and trespassing of property. Consequences of these threats being acted out may result in theft, damage to property or even worst scenarios. Therefore, it can be clear that such monitoring devices are highly important in functioning correctly, and perform the task they are intended to with absolute efficiency. In other words, such devices must perform to a certain standard.

In order to ensure this standard is met, various organisations set up rules and guidelines which specify exactly how such devices must perform. One example is the National Fire Protection Association, which sets out guidelines for fire and smoke alarms testing. As quoted from Richardson and Moore, the guideline "covers the application, installation, location, performance, and maintenance of fire alarm systems and their components (Richardson & Moore, 2000, p. 20)." In other words, this standard covers the exact testing procedures which must be carried out against an alarm system and defines the exact metrics which make the device acceptable for use.

With the increasingly rapid evolution of technology, it has been a lot easier for businesses and organisations to take advantage of local area networks for interconnecting their information systems together. With this growth, there has been increasing popularity in the usage and development of the internet as a whole (Howard, Paridaens, & Gramm, 2001, p. 117). Many computer threats have also appeared, therefore, a need for digital intrusion detection systems. To emphasise this point, the statistics provided in an independent survey of UK businesses conducted by the Department of Business, Innovation and Skills show that in the year 2008, there is still a relatively high number of malicious software and hacking attempts affecting information systems (Figure 1.2.1).

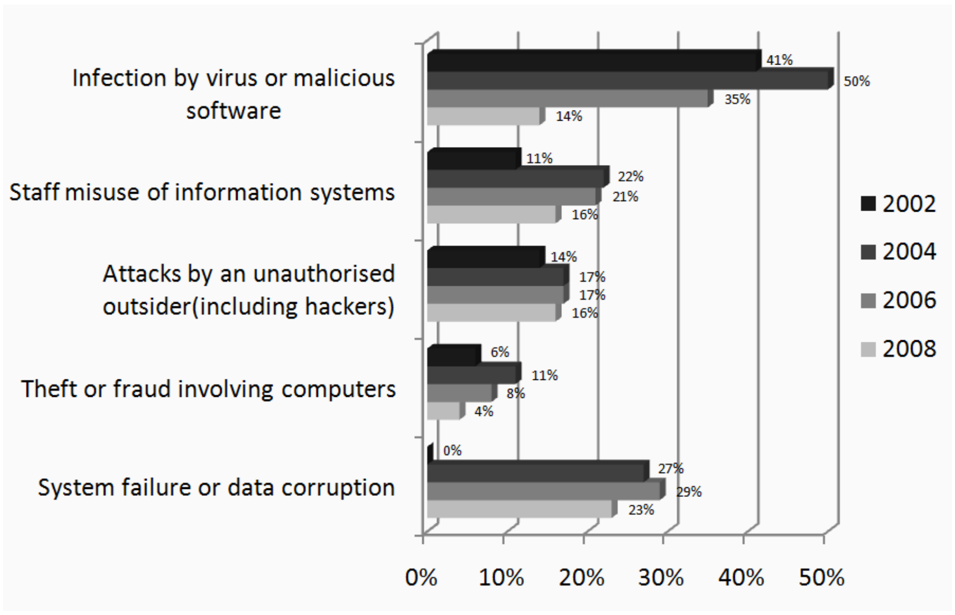


Figure 1.2.1 – Security Breaches in UK Businesses (BIS, 2006), (BIS, 2008)

Although, overall, the survey shows that breaches have been dropping each year, there is still a significant risk of computer threats at this point in time. To elaborate, a much more recent example of a security breach is a case which occurred in November of 2009 when criminals stole over £600,000 from bank accounts by simply installing a Trojan on host computers (Goodin, 2009, para. 1). Many more similar cases and examples of such breaches in information security can easily be found by simply monitoring the news on a daily basis.

Since information security threats have just as serious consequences as fires or trespassing would one may conclude that there is a definite need for digital IDSs to monitor and detect such threats if they occur. The issue at this point in time is that there is no standard method which can be used to assess the effectiveness of these programs (Mell *et al.*, 2003, p.1). In comparison with the standards applied to testing, for example, a fire alarm, the methodology in evaluating a digital IDS is sorely lacking. Thus, it can be seen that it is highly important that exact methodology for testing and evaluating digital IDSs, one in which sets the standard of evaluation, is an absolute necessity.

It should be noted that different categories of digital IDSs exist. This includes Host-Based, Network-Based and Distributed-Based (see Section 2.3.3). The main scope and main focal point of this project will be on Network-Based IDSs (NIDS) though the other categories will be touched upon as well.

1.3 Aim and Objectives

The overall aim of this project is to produce a prototype framework which is capable of carrying out an evaluation of a NIDS. In order to meet this aim, the following three main objectives must be met:

1. Review and research the taxonomy of IDSs and existing methodologies for evaluation including the testing methods applied along with metrics of evaluation.

2. Design a framework which can be used to evaluate a NIDS based on the findings of the literature review with justification for design choices made.

3. Implement and evaluate the framework by testing it against an NIDS to see what results are produced in order to assess the capabilities of the framework.

1.4 Thesis Structure

This thesis is split into six main chapters. They are described as follows:

- **Chapter 1** – Introduction: Provides the project overview and background to the subject of Intrusion Detection Systems. The key aim and objectives of the project are also defined along with the thesis structure.

- **Chapter 2** – Literature Review: An introduction to information security is first provided along with a research on the taxonomy of IDSs. The main focal point of the chapter, an analysis of the existing methodologies used in the evaluation of IDSs is then reviewed with emphasis on analysing the strengths and weaknesses of the method. Finally, a conclusion is reached which provides the driving force behind this project.

- **Chapter 3** – Design: Based on the conclusion reached in the literature review, this chapter proposes a design of a prototype framework which is required for carrying out an evaluation of an NIDS with justifications as to why the approach is viable.

- **Chapter 4** – Implementation: This chapter documents the steps taken in creating the framework based on the design from the previous chapter. Snippets of code along with screenshots are provided to show the many functions of the implementation.

- **Chapter 5** – Evaluation: An evaluation of the prototype framework is provided in this chapter. Using the framework, an experiment, in the form of a set of tests, is carried out against the NIDS known as Snort. The experiment is described along with results produced. An analysis of results is also provided.

- **Chapter 6** – Conclusion: A conclusion to how well this project met the initial aim and objectives originally set out is provided. A self-reflection section is also provided to discuss some of the difficulties faced and how they were overcome along with a discussion on project management. Finally, this chapter ends with a section dedicated to describing some directions in which future work can be taken in this subject area.

2 Literature Review

2.1 Introduction

This literature review first provides a background to information security (Section 2.2). To understand the basic concepts of IDSs a review of the taxonomy and characteristics (Section 2.3) of this topic is then carried out. Some of the main network security threats are then analysed (Section 2.4) in order to understand what an IDS should be capable of detecting.

The main focal point of the review, methodologies in the evaluation of IDSs, is then provided (Section 2.5) along with looking at existing programs and tools which allow for testing of IDSs (Section 2.6). Finally, a conclusion (Section 2.7) is drawn which, based on the methodologies reviewed, identifies three main requirements for an effective evaluation of IDSs.

2.2 Information Security

A clear and concise definition of "security" may be found by Whitman and Mattord (2008), in which they state that it is a form of "protection against adversaries – from those who would do harm, intentionally or otherwise (Whitmand & Mattord, 2008, p.8)". In other words, security is safeguarding against both intentional and unintentional threats. To simplify this even further, we can consider security as a form of protection from danger. Security can be applied to many different areas (such as physical security as an example) but, for the scope of this thesis, the main focus will be on information security.

Information security can be considered an all encompassing term and consist of the following components: management of information security, computer & data security, policies and network security (Whitman & Mattord, 2008, p. 8). In other words, information security is a combination of all four components and applying each area in practice. Perhaps the most important concept behind information security is the key principle of CIA (Confidentiality, Integrity and Availability). This key principle is described in the section which follows.

2.2.1 Confidentiality, Integrity and Availability

In regards to information security, it is widely accepted that a threat to information systems will generally affect one or more of either: confidentiality, integrity or availability of a resource (Whitman & Mattord, 2008, p. 8). Thus, the purpose and goal in information security is to protect these three key principles. The "resource" itself can be anything which is deemed important, whether it is a physical entity (such as paper documents or computer hardware) or a virtual entity (such as a computer database or source code to a piece of software). Confidentiality is making sure information is accessible to authorised persons only; integrity is ensuring data is correct and accurate whilst availability refers to having access to information systems when required (Cavalli, Mattasoglio, Pinciroli & Spaggiari, 2004, p. 298).

To elaborate further, Table 2.2.1 is shown in which gives examples of a threat being against a resource and the principle in which would be compromised. The table provided gives example of three threats. Each of them will compromise one of the principles in information security. Such threats may be grouped into different categories and it is the intention of the next section to look into the different types of threats faced in information security.

Table 2.2.1 – Example of Threats against C.I.A

Resource	**Threat**	**Compromises**
Classified medical record of a patient.	Gaining unauthorized access to this record and reading or distributing the information to others.	Confidentiality
Database which contains detail of a Bank's customer.	Breaking into the database, and changing the data of customer details.	Integrity
A Server hosting a website.	Using computer security exploits to bring the system down.	Availability

2.2.2 Categories of Threats in Information Security

Attempting to provide an exhaustive list of all threats faced by information systems would be both very time consuming and, perhaps, even impossible due to the sheer volume that exist (NIST, 2006, p. 21). Ultimately the threats faced by users or organisations may differ dependant on the environment where the information systems are contained.

However, at the same time, it is important to establish what some of the main threats of this subject is. Therefore, the following paragraphs provide categories of well known threats which are derived from the work of Whitman and Mattord (2008). Additional references have been cited for clarification on some of the threats shown:

- *Human Error*: Involves accidents and mistakes which are made by users of information systems. This may be due to lack of training or inexperience when using information systems (Whitman & Mattord, 2008, p. 42). In some cases this threat may involve a user being tricked by social engineering - which involves exploiting someone through manipulation of their behaviour (Workman, 2007, p. 316) in order to get them to release confidential information such as their password to computer systems.

- *Trespassing:* Involves both intentional and unintentional access to unauthorized data and information. Although such an act may have been unintentional, the result is still a loss of confidentiality in information (Whitman and Mattord, 2008, p. 45). Furthermore, if the act is intentional there is the possibility of it being industrial espionage which is when information is gathered from one company and given to a competing company in order to give them an advantage (NIST, 2006, p. 26). This threat may also be physical, in that a person walking into an unauthorized area would be an example of trespassing.

- *Theft:* Includes both the physical theft of computer systems and equipments along with the theft of data in the form of information.

- *Sabotage*: Forms of sabotage may come from employees who damage or misuse computer equipment (Corsini, 2008, p.8) as well as more serious cases where company websites are vandalised which results in a loss of reputation (Whitman & Mattord, 2008, p. 52). The reason behind such sabotage may include disgruntled employees or simply mischievous acts which were carried out in boredom.

- *Software Attacks*: Includes all forms of malicious software such as worms, viruses and trojan horses. In most cases, systems infected with such software will result in decreased productively from users due to the time and cost required to clean the infected system (NIST, 2006, p. 27). Furthermore, brute force attacks - which involve software which tries to guess the password to a username by attempting all possibilities (Cornsini, 2006, p. 8) - and denial of service attacks - which attempt to starve the system under attack from all resources (Ptacek, Newsham & Simpson, 1998, p. 10) which result in a crash - also fall under this category.

- *Technical Hardware Failure:* Hardware which contains flaws due to design errors or lack of maintenance over time may suffer instabilities or even complete failure. One example of a hardware design flaw is from the Intel Pentium II CPU which contained a floating point bug which resulted in calculation errors (Whitman & Mattord, 2008, p. 62) requiring a total recall of the product hence a great loss in money.

- *Technical Software Failure:* Programming flaws and errors overlooked during the design and implementation of software results in vulnerabilities being exploited or bugs in a programme (NIST, 2006, p. 22). The same may apply if software is not updated over time.

- *Forces of Nature*: Natural disasters such as fires, floods, earth quakes and lightning all pose a threat to information systems. Results from such disasters may include the worst case scenario of systems being damaged beyond all repairs. The unpredictability of natural disasters means it can be difficult to safeguard against such threats (Whitman & Mattord, 2008, p. 59).

Threats such as human error may be reduced through training of users involved in information systems. The chances of technical hardware and software failing is also reduced if proper maintenance and upgrade to components and resources as carried out. However, at the same time, even by applying every safe guard possible, there is always a small risk of a threat being carried out due to unknown vulnerabilities. This is especially true in the case of such threats as software attacks and information theft since no physical evidence may be left behind after the threat occurs therefore users may be completely unaware an attack took place. The point to be made here is that threats cannot always be prevented but, they can be detected, stopped as soon as possible and safeguarded against in the future.

As the first chapter of this thesis has already described, physical detection systems such as fire alarms or C.C.T.V's allow for the detection of specific real-life threats such as fires or trespassers but, there is no possibility in these devices detecting a denial of service attack, a rogue virus spreading itself throughout a computer network or an application suddenly modifying user passwords. Thus, this lead on to the need for mechanism for the detection of non-physical information system threats which may be achieved by deploying digital IDSs. The next section of this report will focus on the taxonomy of this subject.

2.3 Intrusion Detection System Taxonomy

The meaning of taxonomy is in providing a unique label for each entity within a subject area. The purpose of which is to allow researchers to easily communicate and group subjects matters together. To elaborate, the work of Almgren, Lundin & Jonsson (2003) state that having a good taxonomy allows other people to express their thoughts and ideas in a manner which is universally understood by all others working on the same research area (Almgren *et al*., 2003, p. 57).

Unfortunately, a singular and widely agreed upon taxonomy of IDSs does not exist (Tucker, Furnell, Ghita & Brooke, 2007, p. 88-89). This is due, partly, to the fact that as you delve deeper into the technical aspects of IDS, it becomes more difficult and complicated to classify specific elements into a standard category due to ambiguity or technical complexity.

The purpose of this section is not to provide a comprehensive taxonomy into IDSs – such an objective is outside the scope of this thesis. Rather, the focus will be in providing a taxonomy on concepts, terms, and ideas which *is* widely accepted by researchers. Although the introduction to this thesis has briefly described the general purpose of an IDS, this section intends to delve a lot deeper into this subject area.

2.3.1 Definition and Purpose of Intrusion Detection Systems

A standard definition of the term intrusion is described as "the act of intruding or the state of being intruded; especially: the act of wrongfully entering upon, seizing, or taking possession of the property of another (Webster-Merriam, 2009)." From this definition, one may conclude that an intrusion is to enter upon a domain which is considered restricted, regardless of whether the act is considered intentional or not.

In other words, if we apply the term intrusion to the topic of information security, this can be considered simply as another form of a threat being acted out (an attack). Furthermore, it should be noted that an intrusion in this context will threaten to compromise one or more of the three principles of information security: confidentiality, integrity and availability (previous discussed in Section 2.2.1). This idea is backed up in the work of Purcell (2007), in which the author states that IDSs are a control measure (Purcell, 2007, para. 6) whilst the work by NIST (2006) reinforces this by stating that the purpose of IDSs is in "identifying attempts to penetrate a system and gain unauthorized access (NIST, 1996, p. 215)." To describe it in another way, an IDS can be defined as a control measure of which the purpose is to detect threats which put information systems at risk.

It is also important to note what is *not* considered the purpose of an IDS. From the work of Kazienko and Dorosz (2003), they state that security devices such as network logging systems, anti-virus detection/protection, vulnerability assessment tools, firewalls and security/cryptographic systems, although similar to the purpose of IDSs, should not be classified under the same category as IDSs (Kazienko & Dorosz, 2003, para. 3). Such security devices will most likely work together in order to provide for multiple layers of defence in information security, but the general functionality of these devices will differ from an IDS.

To provide a summary to this section, the most important point that should be understood is the fact that all threats, from an Information Security stand point, will compromise one or more of the three key principle areas which are, to repeat: confidentiality, integrity and availability. The purpose of IDSs is in detecting threats which compromise any of these three principles. Devices such as firewalls and network logging systems may achieve the same goals, but their functionalities will differ to that of an IDS.

2.3.2 IDS Framework and Characteristics

Many different types of IDSs currently exist, and, as expected, they will function differently dependant on the design choices taken in implementation. It can be helpful for users wishing to learn about this subject if a common model exists which acts as a form of blueprint which relates to all IDSs. The work produced by Porras, Schnackenberg, Staniford-Chen, Stillman and Wu (1998) recognised this importance, and the result of the authors work is the Common Intrusion Detection Framework (CIDF).

In the CIDF, the IDSs are divided into four main components, which are as follows: event generators (known as “E-boxes”), event analyzers (“A-boxes”), event databases (“D-boxes”) and response units (“R-boxes”) (Porras *et al.*, 1998, para. 21-26). Table 2.3.1 provides a summary as to what the purposes of each of these components are.

Table 2.3.1 – Description of the four components of the CIDF

Component Name	**Purpose of Component**
Event Generators (E-Boxes)	The E-Box is the sensor, which will monitor and obtain activity (such as network traffic) and convert the data it sees into the CIDF format so the other components can interpret it.
Event Analysers (A-Boxes)	The A-Box receives data from the E-box and will analyse it. Method of analysis is dependent on the design of the IDS.
Event Databases (D-Boxes)	The D-Box is simply where the analysed data is stored.
Response Units (R-Boxes)	Based on the analysed data, and, once again, on the design of the IDS, a response will be carried out which may include producing an alert (if the data analysed is considered an intrusion) or the event is simply logged.

To provide for a better visual representation of how these components interact with each other, Figure 2.3.1 is presented which comes from the work produced by Ptacek, *et al.* (1998). It should be made clear that although the diagram labels one of the components as a “Countermeasure (C) Box”, this serves the exact same function as the Response Units (R-Box) described in the previous table.

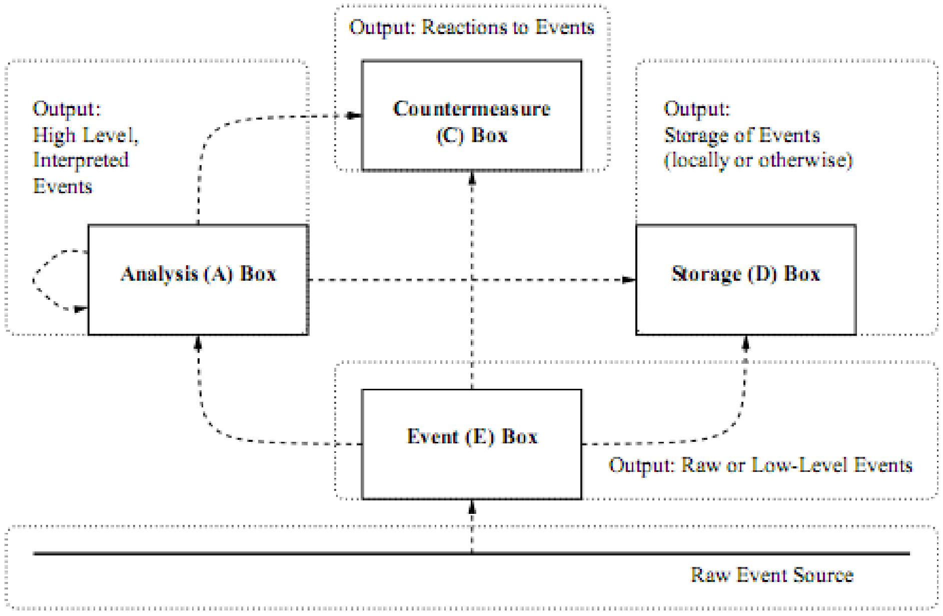

Output: Reactions to Events
Countermeasure (C) Box
Output: High Level, Interpreted Events
Output: Storage of Events (locally or otherwise)
Analysis (A) Box
Storage (D) Box
Event (E) Box
Output: Raw or Low-Level Events
Raw Event Source

Figure 2.3.1 - CIDF interaction method (Ptacek *et al*., 1998)

From the description and the figure provided, it can be seen that each component relies upon each other, both directly and indirectly. What can be summarised is that although different IDSs will carry out the task of intrusion detection using their own methods, all IDSs will still be similar due to the fact that they are based on the design of this framework.

However, criticism has been made against the CIDF. One in particular was made in the technical report produced by the Malicious and Accidental Fault Tolerance for Internet Applications (MAFTIA) project. In this technical report, Powell and Stroud (2001) state that the Response Units (R-Boxes) should not be considered a part of the IDS but rather a separate entity due to the fact that, by definition, the purpose of a IDS is in detecting intrusion, not responding to it (Powell & Stroud, 2001, p. 28). Following on from this statement, the authors have attempted to provide their own refinement to the CIDF which can be seen in Figure 2.3.2. This may be considered a minor piece of criticism though, since both Porras *et al*. (1998) and Powell & Stroud (2001) appear to agree that the response units, regardless of whether or not it is part of the CIDF, is still a requirement since some form of action must take place if a attack is detected.

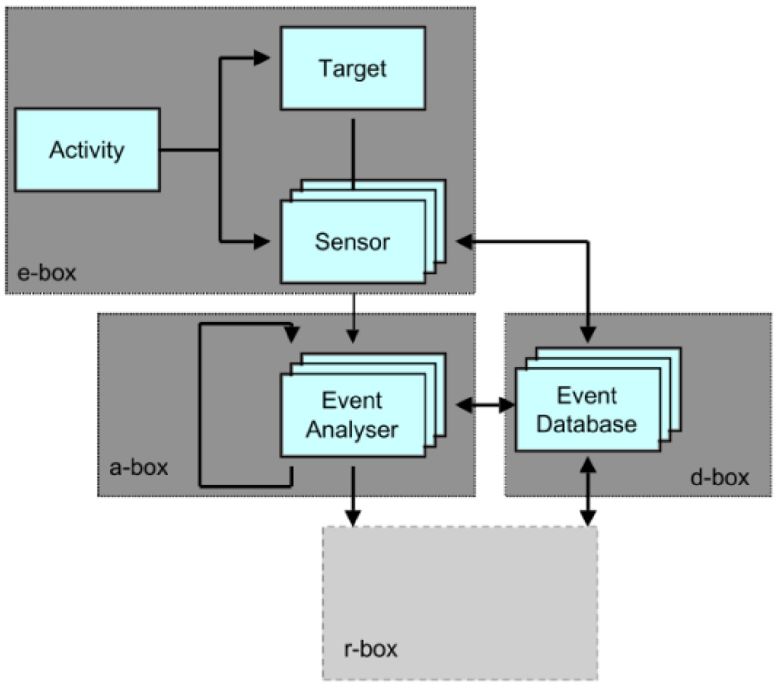


Figure 2.3.2 - Refined CIDF Model (Powell & Stroud, 2001)

Having looked at the basic IDS framework, a review of the main characteristics of an IDS will be carried out. In the work produced by Debar, Dacier & Wespi (1999), the authors defines IDSs as having four main characteristics: Detection Method, Behaviour on Detection, Audit Source Location and Usage Frequency (Figure 2.3.3).

What can be made clear is that each of the three functional characteristics provided in the figure can be easily linked to one of the components of the CIDF described earlier. In regards to the three functional characteristics: **audit source location** refers to how the IDS goes about obtaining activity and converting it to data the system can understand (E-Box), the **detection method** will analyse this data for attacks (A-Box) and **behaviour on detection** defines what action will be taken whether or not attacks are found (R-Box) (Debar *et al.*, 1999, p. 8 - 9).

The usage frequency refers to whether the IDS runs continuously or periodically and since it is a non-functional characteristic no relation can be made with the CIDF. However, to provide for a brief description, continuous monitoring simply refers to an IDS which will monitor for threats without interruption whilst periodic analysis means the IDS monitors for threats on at certain intervals.

The audit source location, including host, network and distributed based, is described in greater detail in Section 2.3.3 whilst information on detection methods, which can be either behaviour or knowledge based, is found in Section 2.3.4.

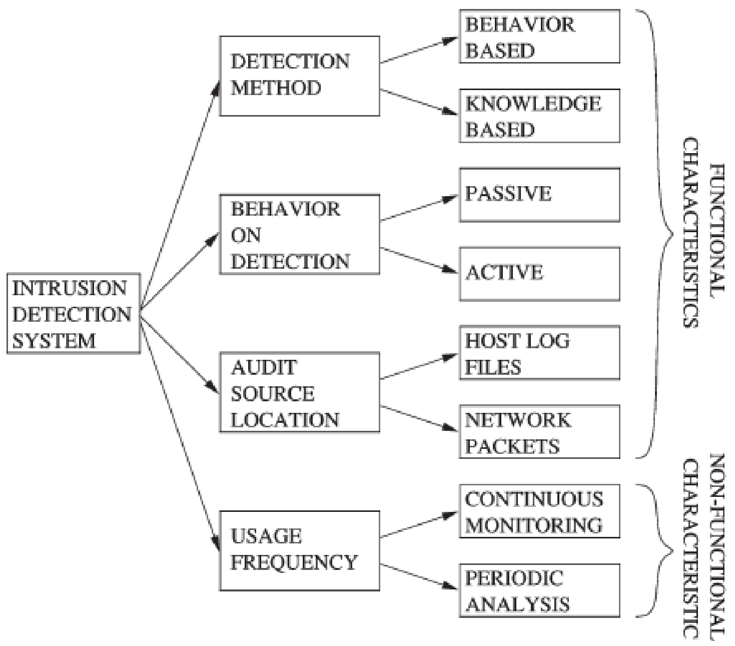


Figure 2.3.3 - Characteristics of an IDS (Debar *et al.*, 1999)

2.3.3 Host-Based, Network-Based and Distributed-Based IDS

The work by Debar *et al.* (1999) states that there are three main categories of audit source locations implemented in IDSs including host-based IDS (HIDS), network-based IDS (NIDS) (Corsini, 2009, p. 15) and distributed-based IDS (DIDS) described by Snapp, Brentano, Dias, Goan, Heberlein, Ho, Levitt, Mukherjee, Smaha, Grance, Teal, & Mansur (1991).

In HIDSs, the internal activities of the computer system in which it resides is monitored and analysed through the usage of audit data which is provided by the operating system (Vigna, Robertson & Balzarotti, 2004, p. 21). For example, an HIDS may monitor the memory usage of programs and produce an alert if the resources taken up by the program begin to fluctuate in an unexpected manner. Figure 2.3.4 shows an example of the location of various HIDSs residing in some of the workstations and one of the servers on a network.

In comparison, NIDSs involve looking at both incoming and outgoing network packets and attempt to find any acts of attacks (Debar *et al.*, 1999, p. 817). In other words, an NIDSs main task is in the analysis of traffic on a network. For a NIDS to work, it must be either connected to a span port on a switch or a device such as a router (Beale, Baker & Esler, 2007, p. 5) in order to capture all traffic on a network. Furthermore, a network interface card (NIC) is obviously needed. Figure 2.3.5 shows an example of one single NIDS connected to the switch of a network. The location of the NIDS allows it to monitor all network traffic activity on this network.

In a DIDS, the techniques of both HIDS and NIDS are used, thus this can be considered a hybrid form of IDS (Debar *et al.*, 1999, p. 817). DIDS functions by using multiple IDSs which monitor both the hosts connected to a network, along with the network activity itself and reports all information captured to a centralized IDS, known as the DIDS director (Snapp *et al.*, 1991, p. 168). Figure 2.3.6 shows an example of the usage of a DIDS on a network. It should be noted that in DIDSs, there is the possibility of an IDS being both a HIDS and NIDS at the same time (Beale *et al.*, 2007, p. 8) which is not shown in the figure. The main point to be made is that all NIDS/HIDS will report to the DIDS director.

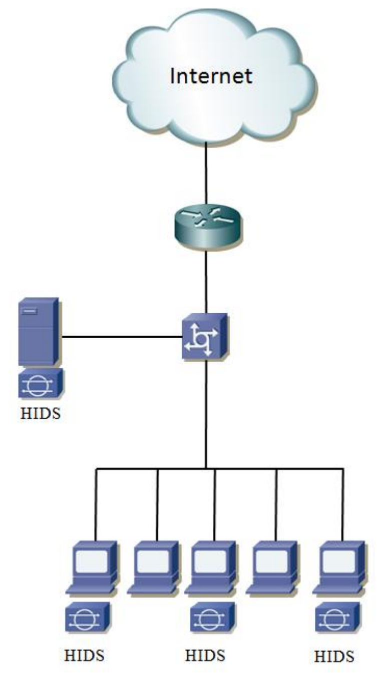


Figure 2.3.4 – Example of HIDSs on a Network

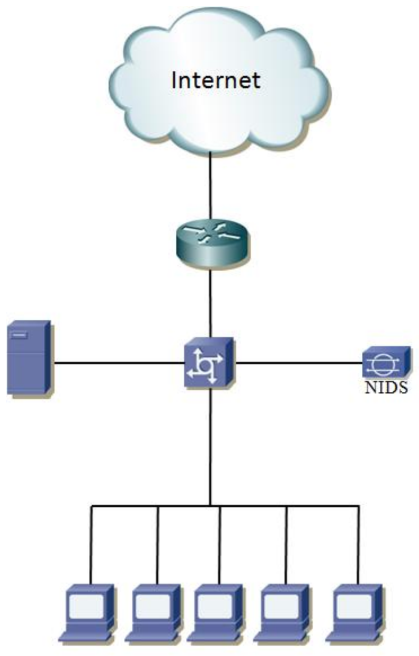


Figure 2.3.5 – Example of an NIDS on a Network

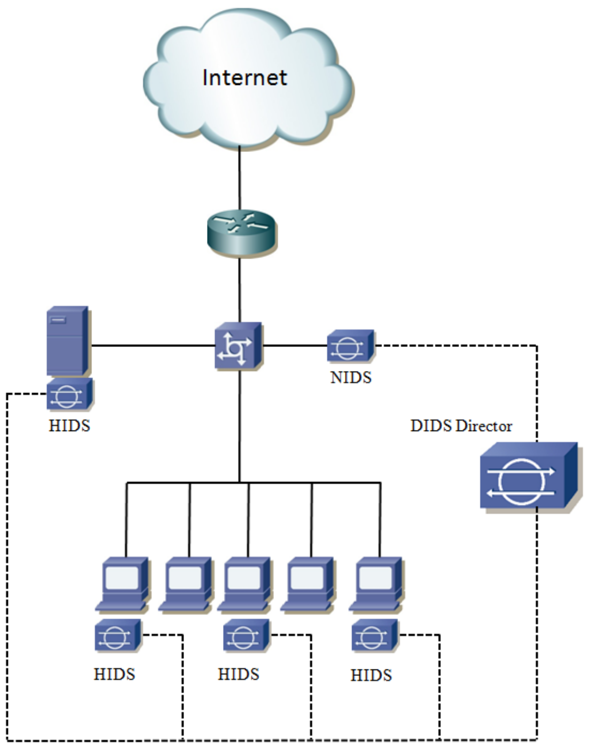


Figure 2.3.6 – Example of DIDSs on a Network

What may be concluded is that there are three main types of IDSs: HIDS, NIDS and DIDS. The use of the term "IDS" is used to describe intrusion detection systems generically, whilst if one was to describe a topic in regards to a specific category of IDS, the acronyms HIDS, NIDS or DIDS will be used instead.

2.3.4 Detection Methods

The work of Debar *et al.* (1999) states that there two main forms of detection methods: behaviour based and knowledge based. These are more commonly known as anomaly detection and misuse detection methods. The purpose of both methods is in attempting to detect any attacks or intrusions on a system. The main characteristic of anomaly detection is in looking for any unexpected changes in behaviour of a system against what is considered "normal" behaviour whilst misuse detection involves comparing incoming threats against a predefined knowledge base in order to decide whether the threat is considered an attack or intrusion (Debar *et al.*, 1999, p. 810).

As mentioned in the above paragraph, an anomaly detection method works by comparing normal behaviour against the current pattern of behaviour in a system. In order to achieve this task, the main challenge in anomaly detection methods is in learning what is considered "normal" behaviour. The work by Axelsson (2000) describes the two main approaches which are used to achieve this goal: self-learning or programmed anomaly detection. In the self-learning approach, the anomaly detection system will begin to automatically monitor events, such as live network traffic, on the environment it has been implemented on and attempt to build information on what is considered normal behaviour (Axelsson, 2000, p. 5). This is otherwise known as online learning (Gong, 2003, p. 3). In the programmed approach, the anomaly system must manually learn what is considered normal behaviour by having a user or some form of function "teaching" the system through input of information (Axelsson, 2000, p. 5). This is otherwise known as offline learning, and may involve feeding the system a network traffic data set which contains normal network traffic (Gong, 2003, p. 3). One of the most common methods for building the behaviour information in both self-learning and programmed anomaly detection systems is through statistical analysis. This involves measuring the average time it takes for a task to be carried out, such as the average time taken for a user to invoke a login to a server, and storing this information as variables (Debar *et al.*, 1999, p. 813). These variables are then used in comparison against the activities when the anomaly system is deployed.

Misuse detection methods, in comparison, require that all known threats will be defined first, and the information regarding these threats to be submitted to the NIDS. Thus, the NIDS is able to then compare all incoming, or outgoing, activity against all known threats in its knowledge base and raise an alarm if any activity matches information in the knowledge base. The information stored in this knowledge base is usually known as signatures. Misuse detection methods normally requires a user to manually define all signatures it should detect, therefore only a programmed approach can be used for this detection method (Axelsson, 2000, p. 6). The process for actually matching a signature with an attack include simple string matching – which involves looking for unique key words in network traffic to indentify attacks – to more complex approaches such as rule-based matching which defines the behaviour of an attack as a signature (Axellson, 2000, p. 7).

Both anomaly and misuse detection methods have their own advantages and disadvantages. In terms of misuse detection, this method is very reliable in detecting attacks so long as the signature has been properly defined in the knowledge base. Furthermore, since the signature must be predefined, if an alarm is triggered, then we can easily determine what type of attack was actually detected by the system. The obvious disadvantage here is that any attack which does not have a predefined signature will never be detected (Zhengbing, Zhitang and Junqi, 2008, p. 2). The opposite is true of anomaly detection methods in which new or unknown attacks can be detected since this approach is based on behaviour rather than predefined signatures however, the main disadvantage is that the system may not provide in-depth details such as the type of attack which raised the alarm and whether or not it was an actual valid attack (Deri, Suin and Maselli, 2003, p. 2).

Finally, as mentioned earlier in this section, when an IDS - using either the anomaly or misuse method - detects an attack, an alarm will be raised. As described by the author Beale *et al.* (2007), the main four types of alarm consist of: true positive, false positive and false negative (Beale *et al.*, 2007, p. 13). A short summary of each alarm is described as follows:

- *True Positive* refers to when an attack has been detected and alerted properly.
- *False Negative* refers to when an attack takes place but no alarm has been raised.
- *False positive* refers to when an alarm is raised but no attack has taken place.
- *True Negative* which refers to when no attack takes place and no alarm is raised.

2.4 Network Security Threats

Since the focal point of this thesis is on NIDSs, some network threats will be looked at in this section. Similar to Section 2.2.2, attempting to compile a comprehensive list of threats is an extremely difficult task, therefore, only the most common and well known network security threats have been included. Theoretically, an IDS should be capable of detecting all threats which are described here.

2.4.1 Surveillance/Probing

The purpose of surveillance/probing is to gather information on a network (Wagh, 2009, para. 1) such as topology, IP addresses of machines and open network ports. Although surveillance or probing carried out on a network is not considered harmful by itself, it is usually a clear indicator that an attack may result in the future (Buchanan, 2009, p. 50).

Network surveillance and probing can be carried out with many types of utilities and programs but, perhaps the most popular one used today is known as Nmap (Lyon, 2009). Surveillance and probing carried out on Nmap is usually known as a sweep or a scan, and many types exist, including: port scan, port sweep and ping sweep. Port scan, as the name will suggest, is in scanning a network host in order to look for any open ports whilst port sweep involves scanning multiple hosts within a network with the same objective in mind. Pingsweeps will attempt to “ping” each host on a network to see whether it is up.

2.4.2 User Privilege Gain

A user privilege gain attack consists of attempts to obtain access to a users account on a system. This usually involves the guessing of user name and passwords using techniques such as brute force dictionary attacks or, in some cases, more sophisticated methods such as social engineering (Workman, 2007, p. 316) may be used instead to trick a user into giving out their password.

An example of a program which can carry such attacks is Hydra (THC, 2009) which is capable of carrying out dictionary attacks (Corsini, 2009, p. 20) on protocols including Telnet, FTP, SSH and HTTP. In dictionary attacks, a user name and password list is supplied to the program along with the target. This program will then attempt to try all combinations of passwords and return a result if one is successful.

Also, related to this threat is user root gains. Once an attacker has managed to gain a user account via tools such as Hydra, attempts may then be made to gain higher levels of access such as the administration or root account (Buchannan, 2008, p. 28). Such an attack being successful could be very dangerous. By having access to the highest level account, an attacker could inevitably have control over an entire system. In other words, the attacker would be free to do as he pleased such as deleting critical system files, browse confidential files or change the password on other user accounts.

2.4.3 Viruses, Worms and Trojans

Viruses, worms and trojans are all considered malicious software. In the work of NIST (2006, p. 27) the following summary of each of the three types of malicious software are described:

- Viruses are a piece of code which attaches itself to an application (such as an .exe file) and is executed when the application is run by the user. Viruses may carry out malicious actions such as overwriting or deleting system files.

- Worms differ from viruses in that it does not require to be attached to an application. Instead, it will exist independently and attempt to replicate itself and propagate throughout a network.

- Trojan horses are disguised as legitimate software, and when executed, will open up a "backdoor" (such as opening up a port) which allows access to that machine from outsiders of the network.

2.4.4 Denial of Service Attacks

The main purpose of a Denial of Service (DoS) attack is in using up all available resources such as CPU and memory (Ptacek *et al.*, 1998, p. 39) on either the target system or even the NIDS which monitors traffic on the target network. This may result in the target system crashing, or simply not responding as expected. In the case of carrying out a DoS attack on an IDS there is the possibility of the system letting additional attacks slip past undetected as it attempts to recover. One of such program which will allow for carrying out of DoS attacks is known as Hping3 (Hping3, 2009).

Various methods may be used in order to carry out a DoS attack. The three main types of attacks consist of TCP floods, ICMP floods and UDP floods (Houle, Weaver, Long & Thomas, 2001, p. 3). An example of a TCP flood involves sending huge numbers of request packets (SYN) to a server in order to starve it of all resources, whilst ICMP may involve simply sending large numbers of pings (echo request packets) to a server so that it consumes all its resources in attempting to reply. UDP floods function similar to TCP floods. It is not always necessary for the attacker to send a flood of traffic to the victim. One such example is the LAND attack, in which a single packet is crafted that connects a victim's network socket to itself. Over time, this can cause a system to lockup.

A variation of a DoS attack, known as a Distributed Denial of Service (DDoS) attack exists. In DDoS the use of multiple computers to send out requests to a single receiver is carried out (Kargl, Maier & Weber, 2001, p. 516). DDoS attacks are usually controlled by one main machine known as the Master program and, when initiated, it will communicate with all other machines which have a "bot" installed on them to initiate an attack on a single target.

2.5 Methodologies in Evaluating IDSs

Many evaluation methodologies have been proposed by researchers over the past few years. The main purpose of this section is in analysing some of the methods used and attempt to gain some perspective on what may be considered an effective methodology. To achieve this goal, the first concept which is reviewed is the two main techniques used in carrying out an evaluation: black-box and white-box testing.

2.5.1 Black-Box and White-Box Testing

Two main techniques may be applied when carrying testing in general: black-box testing and white-box testing. Table 2.2.1 provides a definition of black-box and white-box testing which is summarised from the work of Sharma, Kumar, and Grover (2007).

Table 2.5.1 – Definition of Black-Box and White-Box Testing (Sharma *et al.*, 2007)

Testing Technique	**Definition**
Black-Box	Testing is applied to an entity where the internal workings are unknown.
White-Box	Testing is applied to an entity where the internal workings are known and are changeable.

Although the two testing methods are more commonly seen in software engineering we may still apply such techniques to evaluating IDSs. For example, in the work of Gadelrab and El Kalam Abou (2006), the author states that there are two main methods in evaluating IDSs, both of which follow either the principles of black-box testing or white-box testing. The authors refer to the principles as evaluation by test (black-box testing) and analytic evaluation (white-box testing) (Gadelrab & El Kalam Abou, 2006, p. 271).

An example of analytical evaluation is given in the work of Alessandri (2004). The basis of this approach is in predicting whether or not the design of the IDS will be able to detect specific attacks (Alessandri, 2004, p. 13). In order to do this, the author attempts to group attack types into classifications and provide a description to the system on how each class of attack will behave. By comparing the IDSs design against certain classes of attack, they propose that their method should predict whether or not the attack will be effective. This is considered white-box testing due to the fact that the inner-workings of the IDS must be known in order to carry out the evaluation effectively (Gadelrab & El Kalam Abou, 2006, p. 271). Although this is a viable approach, it is more suited to testing IDSs under development since the inner workings of the IDS must be known before such evaluation may take place (Alessandri, 2004, p. 44).

The other method, evaluation by testing, can be considered a black-box testing technique. The general principle behind this method is that the IDS will be tested against various attacks, preferably in conjunction with background traffic (Gadelrab & El Kalam Abou, 2006, p. 271). After testing is completed, the IDS will be evaluated against certain metrics which are defined by the designer of the test. This is considered a black-box testing technique due to the fact that during the evaluation process, we generally don't care how the IDS will handle the attack, but whether or not it detects the attack, and how efficiently it does so in regards to the metrics defined.

It can be summarised that evaluation by testing (black-box testing) provides for a much more solid basis in terms of evaluation results, since metrics must be defined and accounted for. In comparison, since the results from analytical evaluation are simply predictions, the accuracy of the evaluation may be a issue. At the same time, this does not mean analytic evaluation is irrelevant since this approach will provide for a far more solid design of an IDS which is still under development.

Since evaluation by testing is more relevant to this literature review, the next few sections will look at some of the methods which various researchers have used in evaluating IDSs. Two main categories of evaluation by testing exist: online and offline evaluation. The next section provides a brief summary and difference between these two forms of evaluation.

2.5.2 Real-time and Offline Evaluation

As described by Sommers, Yegneswaran and Barford (2005), offline evaluation consists of "the use of canonical packet traces for offline tests (Sommers *et al.*, 2005, p. 1)". In other words, the basic idea of offline evaluation is in recording network packets, otherwise known as data sets, and then playing the traces back against the IDS under evaluation. By comparison, online evaluation involves testing IDSs using live network traffic which may be generated using traffic load generator (Ranum, 2001, p.6). To summarise, the difference between offline and online evaluation is as follows:

- Offline evaluation involves playing back data sets to the IDS under evaluation. These data sets may be captured with packet sniffer programs and then played back with a tool such as Tcpreplay (Turner & Bing, 2009) which is capable of playing back network data sets at variable speeds.

- Real-time evaluations does not provide the IDS with data sets, instead live traffic is used. Tools and software may still be used to generate the traffic but in this scenario it is live traffic which is being sent through a network rather than data sets which are being replayed. In other words, the network will "react" accordingly to the traffic which is sent.

Both methods of evaluation will be covered in the following sections. For clarification, the DARPA (Section 2.5.3) and the Automated evaluation (Section 2.5.4) are both forms of offline evaluation. Real-time evaluation consists of LARIAT (Section 2.5.5) and Trident (Section 2.5.6).

2.5.3 DARPA Evaluation

One of the first well known evaluations comes from MIT Lincoln Labs, sponsored by the Defence Advanced Research Projects Agency (DARPA), known commonly as the DARPA evaluation. This work is described by Lippmann, Haines, Fried, Korba and Das (2000). The main goal in the DARPA evaluation was to carry out a non biased measurement on the performance of various anomaly-based IDSs along with producing an evaluation data set which could be used by others in testing their IDSs (McHugh, 2000, p. 267).

In order to carry out these objectives, Lincoln Lab set up a test network and, through the use of programs, to emulate a large number of workstations, and scripts, created synthetic background traffic mixed with attack traffic at certain periods of time (Brugger & Chow, 2007, p. 1). The traffic is then captured with a packet capture tool and saved as a data set. This data set could then be played back against the IDS under evaluation to access its performance. Both inside and outside traffic was captured for this experiment. The term "inside" refers to network traffic which would be seen within a local area network (LAN), whilst "outside" would be any traffic outside of this LAN. A router is used in the experiments to separate the inside and outside traffic. Figure 2.5.1 has been presented to show the DARPA test bed which was used.

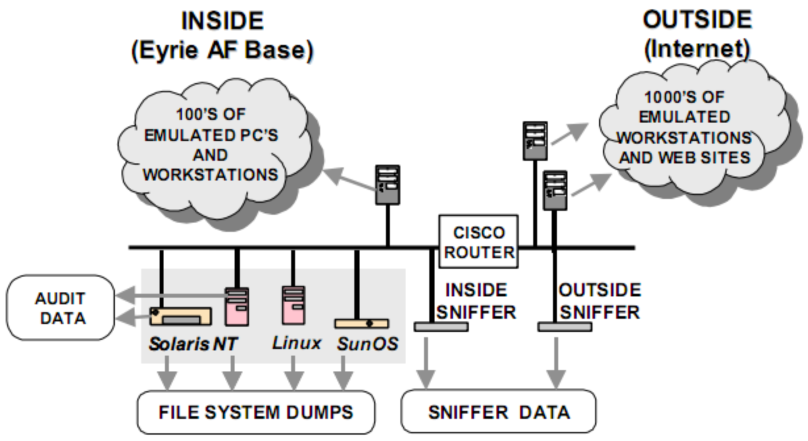


Figure 2.5.1 – DARPA Evaluation Test bed (Lippman *et al*., 2000)

The data sets used in the DARPA experiment was released to the public in 1998 and is referred to as the Intrusion Detection Evaluation (IDEVAL) 1998 data set. The 1998 data set contains 7 weeks of training data, used to train the anomaly system, and two weeks of test data used to test the IDS for detection performance (Brugger & Chow, 2007, p. 2). In 1999, a second data set was produced by the same group, known as the IDEVAL 1999 data set, which contained improvements such as containing stealthier and a wider range of attacks (Brugger & Chow, 2007, p.2). The 1999 data set was made up of two weeks of training data and three weeks of test data (Lippmann *et al.*, 2000, p. 579).

With the use of static data sets, repeatability in experiments is easily achieved and, furthermore, the data set is easy to obtain and free to download. Thus, it may seem that the DARPA method is very well suited for testing IDSs. Unfortunately, both the data sets, and the way in which the evaluation was carried out has faced a lot of criticism by researchers.

In regards to the data set, the IP header attributes contained in the synthesized background traffic including source IP addresses, destination IP addresses, Time To Live value and TCP options all had a small fixed range (Mahoney, 2003, p. 237). Real live traffic, in comparison, will generally contain a vast number of different IP addresses and ranges in their attributes. Therefore, the issue here is that the background traffic in the DARPA evaluation may not be considered similar or realistic enough to the type of traffic normally faced by IDSs.

Criticism has also been made against the attack traffic in the DARPA data sets. During the 1998 DARPA evaluation, four main types of attacks were carried out, which included: User to root, Remote to Local User, Denial of Service and Probe/Surveillance whilst the total number of each attack being carried out is in each of these categories is 114, 34, 99 and 64 respectively (Lippmann *et al.*, 2000, p. 585-586). The flaw here, as stated by Singaraju, Teo and Zheng (2004) is that these attacks - 311 in total – were launched over approximately 9 weeks of testing (5 days per week), therefore the average number of attacks is around 5-6 a day (Singaraju *et al.*, 2004, p. 2). Thus, distribution of attacks may be considered spread too thin in comparison with real life scenarios. The 1999 data set contains a similar problem, with only 200 attacks over 5 weeks of testing (Lippman *et al.*, 2000, p. 579) - this results in approximately 8 attacks per day.

Finally, in the work of McHugh (2000), the author states that the presentation of the evaluation results may not be completely meaningful since only one form of metric is used to determine the performance of the IDS under test (McHugh, 2000, p. 291). This metric which was used is known as the Receiver Operating Characteristic (ROC) curve, which is basically a graphical plot of the number of true positives versus false positives (Ulvila and Gaffney, 2003, p. 453). The problem with using only this singular metric, as stated by McHugh (2000), is that although it shows whether a greater number of true positives or false positives are detected, the ROC curve gives no clear indication to why the IDS under test will behave in such a way (McHugh, 2000, p. 291).

What can be concluded about the DARPA evaluation is that both attack and background traffic lack realism in comparison with real-life network traffic along with the fact that the metric of evaluation used does not give clear indication to the way the IDS performed, regardless of whether it was good or bad. Finally, though it is not mentioned in any cited references, it should be noted that it was more than a decade since the DARPA evaluation, hence newer and more sophisticated attacks may now exist therefore, the validity of testing IDSs using the DARPA attack data sets may not be relevant anymore.

2.5.4 Automated Evaluation

Having seen some of the shortcomings of the DARPA evaluation, it is worth looking at a more recent offline evaluation of IDSs carried out by Massicotte, Gagnon, Labiche, Briand and Couture (2006). In their work, the author's main focus was in providing a more up-to-date attack data set which, similar to the DARPA evaluation, would be made freely available to the public so that others may use it to carry out their own testing against IDSs.

The work by Massicotte *et al.* (2006) consisted of two main elements in which they termed as the Virtual Network Infrastructure (VNI) and the Intrusion Detection System Evaluation Framework (IDSEF). The VNI uses scripts and emulation software to automatically set up a virtual network environment which is used to generate attack traffic against specified targets within the emulated network topology (Massicotte *et al.*, 2006, p. 362). The main purpose of the VNI is in generating attacks on a virtual network and recording the network traffic whilst this take place in order to save it as a data set to be used in evaluating IDSs. Having acquired a data set, this can then be provided to the IDSEF in order to test and evaluate the chosen IDS under test. The VNI uses a layered approach which consists of using programs such as Metasploit (Metasploit Project, 2009) for the generating exploits whilst manipulation of packets for evasion/insertion is carried out using tools such as fragroute (Figure 2.5.2). Unfortunately, a complete taxonomy of all attacks carried out was not provided in this piece of work.

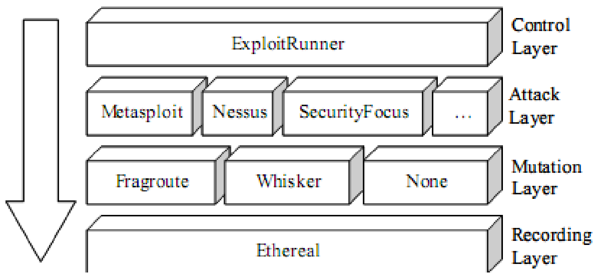


Figure 2.5.2 – Automated Evaluation Attack System (Massicotte *et al.*, 2006)

In order to clarify how this evaluation works, the steps involved in both the setup and execution of the VNI along with the steps involved in using the IDSEF will be described. Ten main steps (Figure 2.5.3) are carried out in the automated evaluation. Steps one to five are summarised from (Massicotte *et al.*, 2006, p. 363) whilst steps six to ten are summarised from (Massicotte *et al.*, 2006, p. 365) and are as follows:

1. *Script Generation* - Using what the author's define as the Vulnerability Exploitation Program (VEP), script generation involves the user defining the types of attacks which should be ran along with specifying the target systems in the virtual network.

2. *Virtual Network Setup* - This step will setup the virtual network dependant on the types of attacks and targets which were specified. For example, if the attack specified was on a web server, then the virtual network that is being setup would reflect this scenario.

3. *Current Attack Script Setup* - Involves configuring the attacks that have been chosen to be executed.

4. *Attack Execution* - Involves running the specified attack(s) against the chosen target(s).

5. *Tear Down* - Involves saving the network traffic which will have been recorded during attack execution into a data set, along with restoring the virtual network to its original state. If additional attacks are to be recorded, then step one to five is repeated, otherwise, the automated evaluation goes to step six.

6. *Data set Selection* - Having acquired all required data sets, the IDSEF will select the relevant attack data sets which should be tested against the IDS which has been chosen to be evaluated.

7. *IDS Evaluator* - These attack data sets will then be provided to the IDS under test.

8. *Next Data set Selection* - Assuming there is more than one attack data set to be test against steps six and seven will be repeated until all data sets have been provided to the IDS.

9. *IDS Result Analyser* - All alarms raised by the IDS in testing each of the data sets will be logged here.

10. *Report Generation* - Finally, a report is generated which shows the types of alarms logged for each of the data sets that were tested against the IDS.

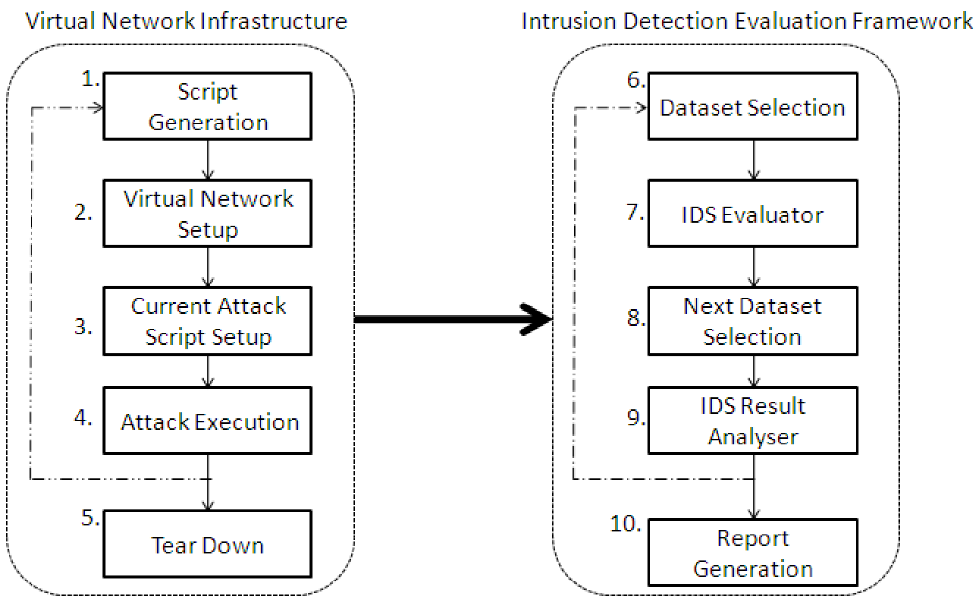


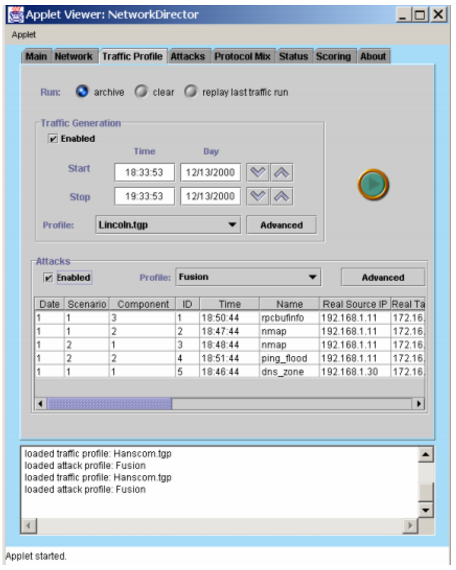
Figure 2.5.3 - Steps Carried out in the Automated Evaluation

The automated evaluation certainly achieves its main goal, which is of providing a more up-to-date data set in comparison with the DARPA data set. Furthermore, the use of virtual machines allows for both ease of automation and repeatability in experiments. However, one main restriction is can be found in this evaluation method. As acknowledged by the authors, no background traffic is generated by the VNI, only attack traffic (Massiotte *et al.*, 2006, p.1). As noted by Mell *et al*. (2003), background traffic is essential to evaluating IDSs since without it, the results will only describe how effective an IDS is in logging true positives with no indication as to what the false positive alarm ratio would be (Mell *et al.*, 2003, p. 14). In other words, an evaluation without background traffic will describe how accurate the IDS under test is in detecting defined attacks, but no information will be provided on the inaccuracy of the IDS – which is whether or not would log attack free traffic as an attack (known as a false positive).

2.5.5 LARIAT Evaluation

The Lincoln Adaptable Realtime Information Assurance Testbed (LARIAT) was designed as a follow up to the 1999 DARPA evaluation (Athanasiades, Abler, Levine, Owen and Riley, 2003, p. 65) and as described by the authors of this evaluation "two design goals were established for LARIAT: (1) support real-time evaluations and (2) create a deployable, configurable and easy-to-use testbed (Rossey, Cunningham, Fried, Rabek, Lippmann, Haines and Zissman, 2002, p. 2672)."

Emphasis on automation of the evaluation was also made by the designers of LARIAT. The justification for this is that manual setup and configuration requirements when testing IDSs can consume a great deal thus, by automating the evaluation process as much as possible ease-of-use and simplicity is achieved for the user carrying out the evaluation (Rossey *et al.*, 2002, p. 2672). In order to achieve this aim, the LARIAT evaluation is implemented on top of a Java applet named NetworkDirector (Rossey *et al.*, 2002, p. 2677). This acts as a "wrapper" in which a GUI interface is built and allows users to interact through the selection of menus and buttons rather than having to manually input commands (Figure 2.5.4).



Run: [✓ archive] clear replay last traffic run

Traffic Generation
[✓] Enabled

	Time	Day
Start	18:33:53	12/13/2000
Stop	19:33:53	12/13/2000

Profile: Lincoln.tgp

Attacks
[✓] Enabled Profile: Fusion

Date	Scenario	Component	ID	Time	Name	Real Source IP	Real Ta
1	1	3	1	18:50:44	rpcbufinfo	192.168.1.11	172.16.
1	1	2	2	18:47:44	nmap	192.168.1.11	172.16.
1	2	1	3	18:48:44	nmap	192.168.1.11	172.16.
1	2	2	4	18:51:44	ping_flood	192.168.1.11	172.16.
1	1	1	5	18:46:44	dns_zone	192.168.1.30	172.16.

```
loaded traffic profile: Hanscom.tgp
loaded attack profile: Fusion
loaded traffic profile: Hanscom.tgp
loaded attack profile: Fusion
```

Applet started.

Figure 2.5.4 – Example screenshot of the LARIAT GUI (Rossey *et al.*, 2002)

The LARIAT evaluation is achieved through the emulation of network traffic (attack and background traffic), which is typically seen in a small organisation connected to the internet (Athanasiades e*t al.*, 2003, p. 65). What this means is that an IDS under evaluation could be placed in between the inside (organisation's LAN) and outside (the internet) network and capture traffic from both sides (Corsini, 2009, p. 38). This is exceptionally effective since it reflects the situation in which most IDSs will be facing in real life scenarios.

User input is required only in the first stage of the LARIAT evaluation. During this first stage, a user is required to select the "profile" required for background and attack traffic (Athanasiades e*t al.*, 2003, p. 64). In regards to the selection of background traffic, this defines the type of environment the evaluation will take place in, along with types of attack free traffic such as HTTP, Telnet, SSH and FTP which should be generated (Rossey *et al.*, 2002, p. 2673). The selection of attack traffic profiles involves a similar process in that the user will select specific attacks which should run against the IDS under evaluation. Upon selecting background and attack profiles, the system will automatically configure the emulated network and begin the process of the testing the IDS under evaluation.

What can be concluded about the LARIAT implementation is that it provides for a very sophisticated evaluation method for IDSs. Not only does it allow users to specify both background and attack traffic profiles, the environment in which the IDS under test is both realistic and in real-time. Unfortunately, there is one major issue LARIAT. As stated by Athanasiades *et al*. (2003), the LARIAT evaluation is part of a United States Government funded project and not available for public use (Athanasiades *et al.*, 2003, p. 66).

2.5.6 Trident Evaluation

The main aim of the Trident evaluation is in allowing for variable mix between benign and attack traffic (Sommers, Yegneswaran and Barford, 2006, p. 1493). In other words, users may specify certain percentages of attack traffic and background traffic. As an example, users may specify a test with 10% attack traffic and 90% benign traffic to see whether the high level of background traffic will have any detrimental effect in the IDSs ability in detecting attacks.

This is achieved using the Malicious Traffic Composition Environment (MACE), written in the Python programming language, which is a tool used for generating attacks on a network (Sommers, Yegneswaran and Barford, 2004, p. 83) and Harpoon, a background traffic generator use to create TCP packet flows and also UDP traffic (Sommers & Barford, 2004, p. 68). Both MACE and Harpoon were developed during separate periods of time but are used in conjunction for the Trident framework (Figure 2.5.5).

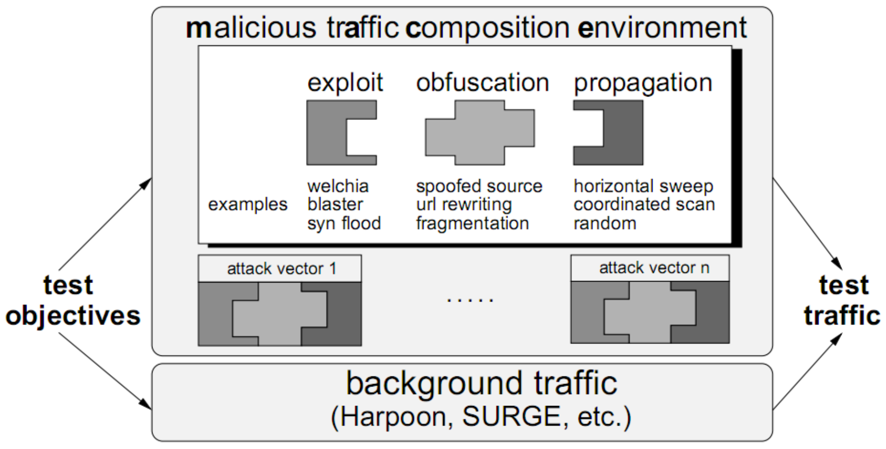


Figure 2.5.5 – Trident Framework (Sommers *et al.*, 2004)

The architecture of MACE consists of four main components: exploit model, obfuscation model, propagation model and background traffic model. The exploit model consists of attacks which target specific vulnerabilities in a system, the obfuscation model is used for insertion/evasion attacks and the propagation model dictates the range of targets (in the form of IP address) to attack (Sommers *et al.*, 2004, p. 83). The forth component, background generation, is optional and used for generating background traffic (in the case of the Trident evaluation, Harpoon is the program used for this purpose). The exploit and obfuscation components are the two main models which define the types of attack capable on MACE, and these attacks are shown in Table 2.5.2.

Table 2.5.2 – Taxonomy of MACE attacks (Sommers *et al.*, 2004)

Host Based					**Network Based**
APPLICATION LEVEL			TRANSPORT LEVEL		
Worms	Back-doors	DoS	Fragmen-tation	Other DoS	
Welchia			rose		
Nimda	mydoom		teardrop1	synflood	smurf
CodeRed2	sdbot	winnuke	teardrop2	pod	fraggle
Blaster			bonk	land	
Dameware			nestea	jolt	
Sasser			oshare		

In the Trident evaluation, the function of MACE is more or less the same in that it is used to generate exploits. However, along with MACE, the Trident evaluation extends the attack database by including the ability to also play back 58 of the attacks used in the DARPA evaluation (Sommers *et al.*, 2006. p. 1493). This is realised in the tool which the authors have named “attack-replay”, and functions similar to Tcpreplay.

Harpoon, the traffic generation tool, consists of two main features: the ability to (1) generate packet traffic at the IP flow level through manual configuration and (2) automatically configure itself to represent the flow of packets in a specific network if NetFlow data is provided (Sommers & Barford, 2004, p. 68).

In regards to the first feature, an IP flow, as defined by Quittek, Zseby, Claise and Zander (2004) is a “set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties (Quittek *et al.*, 2003, p. 3).” The observation point can be devices such as switches and routers, and what Harpoon does is create unidirectional TCP packet flows between these devices to simulate network traffic activity. The second feature of Harpoon is similar to the first, but, instead of users defining the UDP and TCP packet flows, Harpoon may extract NetFlow data logs from routers which would have been previously collected and automatically configure the tool so that the traffic being generated would reflect the scenario previously seen by the router (Sommers & Barford, 2004, p. 69).

As mention earlier in this section Harpoon is used for the generation of benign traffic for the Trident evaluation, but the method used here is a lot more advanced than what the tool could previously achieve. For the generation of benign traffic, Sommers *et al.* (2006) have derived a component named the Automata Generation. This utility first describes the process involved in an application protocol establishing a connection, the exchange of data between client and server is then described and finally, the steps involved in ending the connection is described (Sommers *et al.*, 2006, p. 1492). Network data sets are fed to this utility and individual packets from the same application state will be individually extracted and sanitized to normalise the data. Thus, what results from a large data set will be individual network traces including HTTP, SMTP, DNS, Telnet, FTP and SSH exchanges. Finally, at this point, each of these traces can be given to Harpoon to be transmitted on the network. The automata previously created will be used by Harpoon to play back the traffic.

To summarise the Trident evaluation, the usage of MACE and Harpoon allows for a fine control between benign and attack traffic along with mixing them in a realistic manner. However, one main critique should be made against this evaluation method, especially in regards to MACE. Although a wide taxonomy of attacks has been defined, it should be noted that these are attacks have been specifically coded and crafted for researchers to evaluate IDSs. In other words, what this evaluation may end up assessing is how effective an IDS is, in detecting attacks generated by MACE but not attacks from programs found widely on the internet. There is also a minor critique on the traffic generator, Harpoon. From the work of Corsini (2009), the author states that Harpoon lacks realism since it always uses the same port number for communication (Corsini, 2009, p. 37), an attribute which would not occur in real life network traffic. However, it should be noted that it is unclear whether this problem exists in the Trident evaluation.

2.5.7 Metrics for Evaluation

The term metrics relates to a form of measurement which is used to assess the performance of an IDS after it has been put through testing (Ranum, 2001, p. 2). Many metrics have been suggested, and are used, for the evaluation of IDSs.

In the work of Fink, Chappell, Turner and O'Donoghue (2002) the author attempts to categorise metrics into three main classes: Logistical, Architectural and Performance (Fink, *et al.*, 2002, p. 5-6). In terms of logistical metrics, this relates to how easy the IDS are to setup, maintain and implement into the environment it is intended for. Architectural metrics relates to what the actual intention of the IDS is such as whether it is a host-based IDS or network-based IDS along with the method in which it detects intrusions such as misuse or anomaly detection. In terms of performance metrics, this includes the interoperability between IDS and firewalls/routers along with observed false-positive and false-negative ratios.

What should be noted in the metrics defined by Fink *et al*. (2002) is the fact they appear much more suited for usage from an administrative and managerial perspective when it comes to choosing what type of IDS should be implemented in an organisation or company. Furthermore, as noted by the authors themselves, some of the metrics defined may be quite difficult to measure (Fink *et al.*, 2002, p. 8). For example, how does one go about defining a metric which is able to measure the interoperability between IDSs and firewalls/routers? Furthermore, what type of metric will allow a tester to measure the ease of setup of an IDS which results in a reliable score each and every time?

More relevant is the work of Sommers *et al.*, (2005) in which the authors evaluated IDSs using two main metrics: efficiency and effectiveness. Efficiency is a measure of false-positives whilst effectiveness is a measure of false-negatives (Corsini, 2009, p. 23). Figure 2.5.6 demonstrates the formulae used to work out each metric. In both equations, the calculated result closest to 1 is always better.

$$\text{Efficiency} = \frac{\text{TruePositives}}{\text{AllAlarms}}$$

$$\text{Effectiveness} = \frac{\text{TruePositives}}{\text{AllPositives}}$$

Figure 2.5.6 - Formula for working out Efficiency and Effectiveness Metric (Sommers *et al*., 2005)

CPU utilization, memory usage and packet loss are also monitored. In the case of CPU and packet loss metric, the purpose is to see what relationship either one will have against variable traffic flows (Sommers *et al*., 2005, p. 10). The idea of packet loss being an important metric is further elaborated in the work by Graves, Buchanan, Saliou and Old (2006). As noted, the omission of packets means less information will be provided for forensic systems security as a form of evidence logging (Corsini, 2009, p. 23). In other words, if packets are lost by the IDS, there is less evidence of an attack taking place since no log of the attack may exist.

In a slightly more recent piece of work, similarities on the metrics defined for evaluating IDSs can be found in the work by Gadelrab and El Kalam Abou (2006) in comparison with Sommers *et al*. (2005). In the work of Gadelrab and El Kalam Abou (2006), the authors attempt to split up evaluation metrics into two categories: detection related metrics, and resource utilization metrics. Detection related metrics are used to assess how well particular components of a IDS function, whilst resource utilization relates to what system impact the IDS will have whilst running – in other words, performance metrics. Furthermore, the detection related metrics are further broken down into two subgroups known as macroscopic and microscopic detection metrics (Gadelrab & El Kalam Abou, 2006, p. 273-274). Macroscopic group relates to assessing how well the IDS performed in detecting attacks overall, whilst microscopic relates to assessing how well the IDS performed in detecting individual attacks. Table 2.5.3 shows the metrics suggested in this work and how they are calculated.

Table 2.5.3 – Detection and Resource-Utilization Metrics (Gadelrab & El Kalam Abou, 2006)

Detection Related Metrics	**Definition**
	Macroscopic:
Detection Ratio	DR = (Number of detected attacks / Total number of attacks included in data set)
False Alarm Ratio	FAR = (Number of generated false alarms / total number of generated alarms)
	Microscopic:
Detection Ratio per attack Type	(Number of Detected attacks of a particular type / total number of attacks of this type)
False Alarm Ratio per Attack Type	(Number of generated false alarm for particular attack type / total number of generated false alarms)
Captured Events / Non Detected Attacks	Number of undetected attacks whose events are captured / Total number of undetected attacks
Non Captured Events / Detected Attacks	Number of attacks who events were not captured / total number of undetected attacks
Intrusive Events Drop Ratio	(Number of non captured intrusive events / Total number of intrusive events)

Resource-Utilization Metrics	
CPU Utilization	Percent of CPU used by IDS
Memory Utilization	Percent of memory (RAM) used by IDS

From table presented, we can see that some the metrics defined by Gadelrab and El Kalam Abou (2006) are quite similar to the ones defined by Sommers *et al*. (2005). For example, resource-utilization metrics of CPU and Memory are exactly the same. However, one noted issue with the detection metrics defined by Gadelrab and El Kalam Abou (2006) is the Detection Ratio metric. This is the number of detected attacks divided by total number of attacks included in the **data set**. In scenarios whereby a data set of attacks is not used, this metric would be relatively difficult to calculate since live traffic is unpredictable. However, the work of Gadelrab and El Kalam Abou (2006) does provide for clearer distinction between the different types of performance metrics by sub-dividing them into two main categories: Detection Related Metrics and Resource-Utilization Metrics.

As a summary to this section, what should be apparent is the fact that a quite a large number of metrics have been defined by various authors for the evaluation of IDSs. One very good point was made by Ranum (2001) in which he states "Knowing what not to measure is sometimes a harder problem than known what to measure (Ranum, 2001, p. 3)." To put it another way, the most important factor when it comes to defining metrics is to ensure that they are relevant and they provide a meaningful purpose for the testing which will be carried out on an IDS. With this in mind, it is believed that there is no correct or comprehensive set of metrics which will work for all forms of evaluation of IDS. Instead, the metrics which are defined is dependent on the testing, objective and goal of the evaluation.

2.6 IDS Testing Related Tools and Programs

There are existing tools and programs which are designed can be used for carrying out evaluation of IDSs. These tools and programs generally focus on testing of NIDSs. Such works include Nidsbench created by Anzen Computing (1999), IDSwakeup created by Aubert (2002) and the Metasploit framework created by Metasploit Project (2009).

It must be noted that although these tools and programs may be used to evaluate an IDS, the actual process of the evaluation along with the metrics of evaluation is entirely up to the end user. In other words, they do not represent a comprehensive evaluation methodology of any kind. However, at the same time, it is still worth providing an analysis in order gain an understanding of some techniques which may be employed to test IDSs.

2.6.1 NIDSbench

In the work of Anzen Computing (1999), the NIDSbench toolkit is presented with the purpose of testing NIDSs using two main programs: tcpreplay and fragrouter. Being a toolkit, users are required to specify and provide their own data sets along with manual configuration of fragrouter which is used for the purpose of creating invasion/insertion attacks against an IDS. Due to it being released in 1999, this toolkit is exceptionally out of date since the successor to fragrouter, known as fragroute (Song, 2009), is now actually part of the Tcpreplay package therefore NIDSbench can be considered redundant.

2.6.2 IDSwakeup

IDSwakeup is a tool created by Aubert (2002) which, using Hping2 (an earlier version of Hping3), allows for generation of attacks including various types of DoS attacks. Furthermore, due to the capabilities of Hping2, packet cans be also crafted to mimic more complex attacks. As an example, a packet may be crafted to mimic a HTTP_GET command which could be an indication of an attacker attempting to perform some exploit on a website. The main advantage in the use of this tool is that the attacks are predefined therefore, users can simply specify the target to invoke attacks on and IDSwakeup will take care of the commands to run using Hping2 to create the attacks. The obvious limitation is that the tools purpose is in generation of attacks therefore no background traffic element exists.

2.6.3 Metasploit Framework

The Metasploit Framework is part of the Metasploit Project (2009). This tool is quite similar to Trident, described in Section 2.5.6, in that it provides for libraries of attacks (Sommers *et al.*, 2005, p. 2). One area in which Metasploit could be considered superior to the attacks generated by Trident is that it is open source therefore it may be updated and maintained far more frequently. Furthermore, another advantage in the Metasploit Framework is that a graphical user interface is provided to the user. This allows for ease of automation for the user, since they do not need to enter commands manually, when carrying out testing along with ease of use. Of course this framework is limited to exploit generation only therefore a methodology is still required for effective evaluation of IDSs.

2.7 Conclusion

This chapter has met the main aim of analysing the methodologies used in evaluating IDSs along with providing background to subjects related to this topic. Section 2.5.3 reviewed the offline DARPA evaluation which, although provided significant advances in the research of evaluating NIDSs, is unfortunately flawed due to having a lack of both realism in network traffic and meaningful evaluation metrics.

In the Automated evaluation, reviewed in Section 2.5.4, emphasis was made on creating and capturing attack data sets by emulating network topologies through the usage of virtual machines for offline evaluation of NIDSs. The ease of repeatability in this evaluation along with automation of tasks are the main advantages, but the lack of background network traffic means the evaluation does not closely reflect real world networks.

Section 2.5.5 looks at the real-time evaluation named LARIAT which proves ideal having both realistic attack and background traffic but, it is not available for public use. This is amended in the Trident evaluation, Section 2.5.6, in which both attack and traffic generation tools (MACE and Harpoon) are available to the public as long as (in the case of MACE) there is a legitimate purpose for using them. However, it was also demonstrated that Harpoon may not provide for realistic background traffic and the tool MACE does not provide for surveillance/probing attacks in its taxonomy.

It can be seen that current existing methodologies that are used for evaluation IDSs all contain some form of shortcoming. Ideally, the LARIAT evaluation may actually have achieved a solid methodology, yet the limitation here is that it is unavailable for public use. However, it should be acknowledged that many good points have been made from the methodologies which have been reviewed. In terms of the DARPA evaluation, the critique carried out by McHugh (2003) has highlighted the need for both realistic attack and background traffic. This is again emphasized in the work by Sommers *et al.* (2005) during the Trident evaluation. The work carried out by Rossey *et al*. (2002) has made a very good point in that an evaluation methodology should be as automated as much as possible along with providing ease of use through the implementation of a GUI wrapper. The idea of automation is further backed up in the Automated evaluation carried out by Massicotte *et al*. (2006) in which they use scripts to automate the evaluation process.

It must also be emphasized that the definition of meaningful evaluation metrics is highly critical. Having the wrong or limited evaluation metrics could easily mean a flawed evaluation methodology which was shown in the DARPA evaluation by McHugh (2003). To briefly reiterate from the conclusion of Section 2.5.7, there is no "correct" metrics, only metrics which have meaningful relevance to the purpose of the evaluation.

To conclude, if we are to focus specifically on the strengths of each evaluation methodology, along with taking into consideration the importance of defining the correct evaluation metrics, it can be summarised that a methodology in evaluating IDSs must consist of the following requirements:

1. Inclusion of Realistic Attack and Background Network Traffic
2. Ease of Automation
3. Inclusion of Meaningful Metrics for Evaluation

The main aim in this project is to produce a prototype framework which is capable of carrying out evaluation an NIDS. Based on the conclusion reached in this literature review, a framework must include all three requirements which have been listed to provide for an effective evaluation of NIDSs. The next chapter will describe such a framework along with the design choices required to evaluate an NIDS.

3 Design

3.1 Introduction

The literature concluded that an effective methodology in evaluating IDSs must consist of the following three main requirements:

1. Inclusion of Realistic Attack and Background Network Traffic
2. Ease of Automation
3. Inclusion of Meaningful Metrics for Evaluation

This chapter outlines the design of a prototype framework which attempts to meet all three requirements for the evaluation of an NIDS. The proposed framework allows for a black-box evaluation of the signature-based NIDS known as Snort whilst attempting to follow the three requirements listed as rigorously as possible.

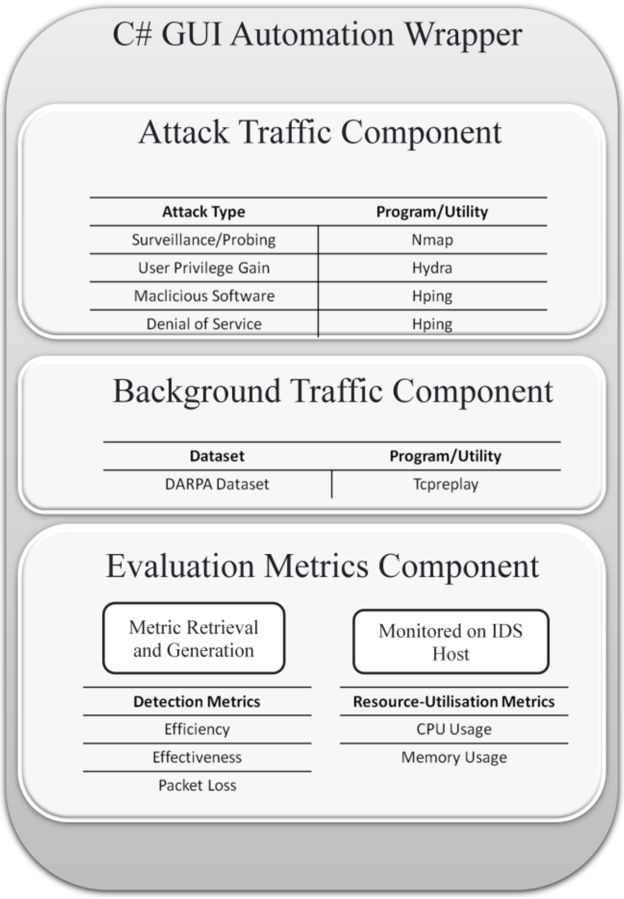
Snort has been chosen since it is freely obtained and widely supported. Furthermore, a huge amount of predefined rules are available for download from Sourcefire (2009) produced by the Sourcefire Vulnerability Research Team (VRT). This makes it a lot simpler to test against, since there is no need to craft user-defined rules in order to test for false-positives. The methodologies reviewed in the literature, including works by Sommers *et al.* (2004); Sommers *et al.* (2006) and Massicotte *et al.* (2006) all performed testing against Snort. Thus, it can be said that this IDS is perhaps the most popular choice among researchers making it viable for evaluation in this project.

Section 3.2 provides an overview of the prototype framework, along with a brief summarisation on design choices made. Furthermore, a schematic of how such the framework functions is provided. A more detailed description of the design choices is then described. Section 3.3 looks in-depth at the design for the inclusion of attack and background network traffic whilst Section 3.4 provides a solution for the ease of automation. Finally, Section 3.5 defines the metrics for the evaluation. Justification for the design choices made is provided in each section. Finally, a conclusion to this chapter is provided in Section 3.6.

3.2 Prototype Framework Overview

As shown in the introduction to this chapter, there is a need for both realistic attack and background traffic when evaluating an IDS. This is especially true in the case of evaluating a NIDS, since the whole purpose of such a system is to monitor network traffic (see Section 2.3.3). The second requirement of automation is also of importance. Automation allows for ease of use for the user, in terms of configuration, along with repeatability in experiments. The third requirement, defining the correct metrics for evaluation, is critical as it allows the user to come to a meaningful conclusion after evaluation of an IDS is complete.

Based on the three requirements a framework (Figure 3.2.1) is presented along with a schematic on how it should function (Figure 3.2.2). Three main components are specified: the attack traffic component, background traffic component and evaluation metrics component. Furthermore, a GUI element is shown which wraps all these components into a single application.


C# GUI Automation Wrapper
Attack Traffic Component
Background Traffic Component
Evaluation Metrics Component
Metric Retrieval and Generation
Monitored on IDS Host

Attack Type	Program/Utility
Surveillance/Probing	Nmap
User Privilege Gain	Hydra
Maclicious Software	Hping
Denial of Service	Hping

Dataset	Program/Utility
DARPA Dataset	Tcpreplay

Detection Metrics
Efficiency
Effectiveness
Packet Loss

Resource-Utilisation Metrics
CPU Usage
Memory Usage

Figure 3.2.1 – Prototype Framework

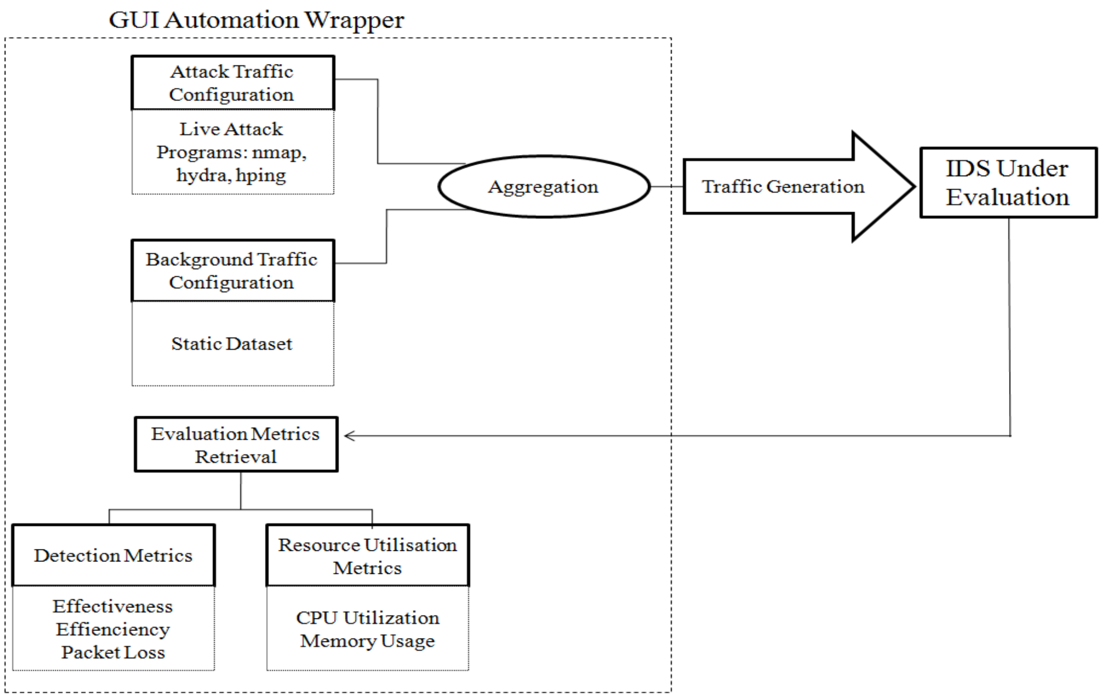


Figure 3.2.2 – Abstract Schematic

In regards to the attack traffic component, it was decided that the usage of publicly available programs should and tools would be put to use in order to generate attack traffic in a real-time manner. It is believed that this will provide for the most realistic scenario in testing an IDS. Some of the tools which are used for this application were described in Section 2.4 and include: Nmap, Hydra and Hping3 for the generation of attack traffic. In the background traffic component, attack free data sets are used to generate benign traffic. Live background traffic is both difficult to manage and control in testing environments, therefore, the use of attack free data sets – to be played back using the tool Tcpreplay – will provide ease of repeatability and predictable flow of traffic.

For the evaluation metrics component, the categories of Detection and Resource Utilization metrics, as defined by Gadelrab and El Kalam Abou (2006), will be monitored. This includes the efficiency, effectiveness, packet loss, CPU utilization and memory usage metrics which were originally defined by Sommers *et al.* (2005). Retrieval of logs produced by the IDS under evaluation and generation of detection metrics are also part of this framework. Resource-utilisation metrics will be monitored directly by the user on the IDS host.

Finally, in order to meet the requirement of automation, a Graphical User Interface is developed which “wraps” the three components previously described together to form the framework as a whole. Thus, rather than have a user individually program, configure and run each component independently, a GUI with the relevant options will be provided as the front end of this application framework so that evaluation of a IDS may take place through a simple click of a few buttons. The GUI is implemented in Visual Studio 2008 and the programming language C#.

A broad overview of this prototype framework has been given. The next section will provide a much more detailed description on the design along with justifications for the choices made.

3.3 Attack and Background Traffic Component Design

In terms of Attack Traffic component, it has been decided that the use of live attacks will provide for the most realistic scenario in testing an IDS. The literature review provided description of the most common network threat which, to briefly reiterate, include threats of surveillance/probing, user privilege gain, DoS attacks and malicious software (worms, viruses and Trojan horses).

In each of the techniques, with the exception of malicious software, an example program has been given which will allow for the generation of live attacks in regards to that network threat category. Due to the security issues which may arise from using real malicious software, the tool Hping3 will be used to craft packets which reflect these threats instead. This packet crafting technique is both similar and influenced by the implementation carried out in the NIDSWakeup tool. However, the design here differs in that Hping3 is used for the crafting of malicious software, a feature which NIDSWakeup does not have.

The Metasploit Framework was considered for generation of attack traffic. However, it was decided that it was not feasible to implement. Although it is open source, it was assessed that the time required to analyse, understand and integrate the code would be too complex a task given the time available for this project.

Through the use of readily available tools and programs, the most realistic attack traffic possible can be achieved since it will be generated in real-time. Table 3.3.1 provides a list of attacks which will be part of this framework along with the tools and programs which are used to generate the actual attacks.

Table 3.3.1- Attacks Included in Application System

Category	Tool/Program to be used
Surveillance/Probing	Nmap
User Privilege Gain	Hydra
Malicious Software	Hping3
Denial of Service	Hping3

Regarding the Background Traffic Component, achieving realistic and controlled generation of benign traffic is still very much an open issue. The closest available tool which will allow for realistic generation of background traffic is found in the work of Sommers *et al.* (2005) using Harpoon. Unfortunately, it was dismissed as not being realistic enough due to using the same port in every connection by Corsini (2009). Another factor, which should not be considered a limitation but more of a hindrance is that getting Harpoon to run requires a lot of effort by the end user. NetFlow logs may be used but in the case of them not being available users must manually create their own topologies and configurations.

Therefore, due to such factors, it was decided that this framework would use attack free data sets for playback of background traffic through Tcpreplay instead of Harpoon. Two main justifications exist for this choice. Firstly, the ease of repeatability will be achieved in playing back the traffic (since the data set will never change in each test) and it will also allow for variable playback speeds in order to assess whether the IDS's performance in regards to the metrics defined will be detrimental in the case of high traffic volumes. Of course, the limitation here is that the only publicly available data set is the ones released by DARPA meaning any evaluation carried using this framework at this point in time is restricted to the DARPA data set.

It is acknowledged that the literature review has dismissed the DARPA data set as being unrealistic. However, with no other option, this is the only choice which remains. But, what must be made clear here is that this framework is not limited to only the DARPA data set. As the tool used to generate background traffic is Tcpreplay, users are free to provide their own data sets to use instead. Furthermore, it leaves the framework open to extension since it may take advantage of newer more realistic data sets, if one is ever released, in the future without any need to modify the design of the framework.

3.4 Automation Wrapper Design

Automation is a highly desirable attribute in evaluation of IDSs as demonstrated by various methodologies carried out in the literature review including Rossey *et al.*

(2002) and Massicotte *et al.* (2006). To justify, consider the example of using Tcpreplay to playback background traffic. A user must first prepare a cache file of the data set then rewrite the traffic (which may consist of both IP and MAC addresses being rewritten) and then finally define the actual playback options. All these tasks must be carried out in command line which adds to both the difficulty and confusion in using such a tool.

Thus, in order to meet the requirement of automation, this framework wraps all programs and functions around a Graphical User Interface. Therefore, rather than having to enter commands individually, a user may be presented with an application where they simply need to select the required options under forms and buttons. Once all configurations have been made, a simple click of a button will allow for generation of traffic (attack and background) to be sent to the IDS under evaluation.

The use of the programming language C# is used to construct a Windows Form GUI. As almost all programs which are part of this framework will require a Linux/Unix environment the application developed in C# will runs using Mono (Mono Project, 2009) which is an open source .NET compatible Common Runtime Language (CLR) designed for the Linux/Unix/Mac OS X/Solaris/Windows platforms.

3.5 Evaluation Metrics Component Design

As the literature review demonstrated, choosing the right metrics to evaluate an IDS is critical in order to assess it in a meaningful way. Taking this into consideration, this framework employs the metrics of efficiency and effectiveness as defined by Sommers *et al.* (2005) along with CPU Utilization and Memory usage metrics. Furthermore, packet loss, described by Graves *et al.* (2006) is an important metric, as demonstrated in the literature review, hence its inclusion. If we apply the categorisation of metrics, as defined by Gadelrab and El Kalam Abou (2006), the metrics of efficiency, effectiveness and packet loss can be considered Detection Metrics, whilst CPU and Memory usage can be grouped as Resource Utilisation Metrics (Table 3.5.1).

Table 3.5.1 – Evaluation Metrics Design

Detection Metrics	**Description**
Efficiency	True-Positives / All Alarms
Effectiveness	True-Positives / All Positives
Packet Loss	The number of packets lost, as reported by the IDS
Resource Utilisation Metrics	
CPU Usage	Percentage of CPU used
Memory Usage	Percentage of memory used

Implementing the efficiency and effectiveness metrics allows for the monitoring of both false-positive and false-negative ratios. In other words, we can assess whether the IDS under evaluation will trigger alerts against benign traffic (false-positives) and whether it will miss real attacks (false-negatives). To perform these calculations, a baseline of expect alarms must first be established. Users will be required to manually run individual attacks against the IDS to assess what this baseline result will be for each attack.

Monitoring of packet-loss allows for assessing whether the IDS can successfully monitor all traffic even in high throughput. The CPU and Memory of the system in which the IDS resides is also monitored to see whether higher resource utilisation results in any detrimental effect to the IDS (such as detection of attacks).

For the purpose of automation, this framework allows automated retrieval of logs produced by the NIDS Snort using a Secure Shell (SSH) connection. All required logs are copied to the machine in which the framework resides and Detection Metrics are automatically generated based on the results parsed in these logs.

One limitation in the current design of the framework is that Resource Utilisation metrics will still need to be monitored manually. This may easily be achieved in multiple ways. For example, on the Windows operating system, the Windows Task Manager will provide a user with ample information in regards to both memory and CPU usage. The same applies for Linux/Unix environments, with tools such as *top* and *vmstat* which generally achieve the same functions as the Windows Task Manager but is command-line driven.

3.6 Conclusion

The main goal in this chapter is to present a framework for the evaluation of NIDSs along with a detailing the design choices made which would adhere to the three main requirements required for a solid methodology in evaluating IDSs. The three requirements consist of:

1. Inclusion of Realistic Attack and Background Network Traffic
2. Ease of Automation
3. Inclusion of Meaningful Metrics for Evaluation

This chapter provided an overview of the framework along with the schematic on how it functions. Furthermore, design choices and justifications were made for each of three components of the framework.

The framework allows for the inclusion of realistic attack traffic generation by making use of existing programs to carry out live attacks whilst background traffic is achieved by using Tcpreplay and a user provided data set. The metrics of efficiency, effectiveness, packet loss, CPU utilisation and memory usage are all part of the design of this framework which will allow for meaningful comparisons to be made such as whether higher resource utilisation results in any detrimental effect against detection of attacks. By wrapping all functions into a GUI application created using Microsoft .NET C# and running it through Mono, automation of carrying out the evaluation may be achieved.

4 Implementation

4.1 Introduction

The design of a prototype framework for the evaluation of an NIDS was described in the previous chapter. This chapter will show how the framework was actually implemented. Snippets of coding are provided to demonstrate the functions carried out by the framework. As the previous chapter showed, three main components of the framework exist including:

- Attack Traffic Component
- Background Traffic Component
- Evaluation Metrics Component

On top of these three components, a GUI has been built which wraps all functions into a singular application. The purpose of this GUI is to provide automation of carrying out the evaluation process along with providing ease-of-use. To provide for a brief overview of the implementation, the next section will first focus on providing a description of this GUI.

4.2 GUI Implementation

The following figure presents a screenshot of the completed GUI (Figure 4.2.1). As the figure presented shows, the three main components to this framework include the attack traffic component, background traffic component and the evaluation metrics component. Each of the three components is clearly separated with the use of tabs. Within each component, multiple subcategories exist which are once again separated by tabs. These subcategories offer the user configuration options along with clearly marked buttons with purposes including invoking attacks, as the figure shows, or inputting configuration changes.

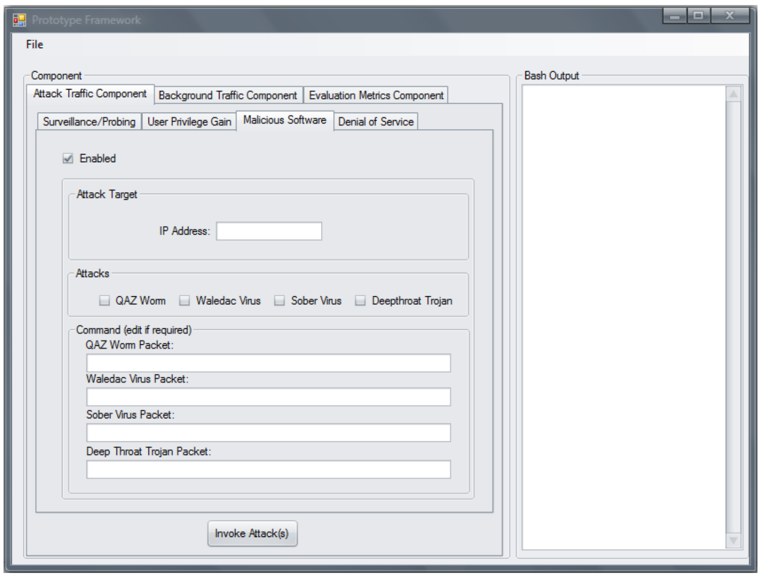


Figure 4.2.1 - Screenshot of Framework

To the right hand side of the application, a Bash Output textbox is presented. Bash is the shell scripting environment used by many Linux distributions (Newham and Rosenblatt, 1998, p. 4) and the purpose of this textbox is to display any output which is redirected from the Bash shell. This allows us to be aware of any information which is produced when running the application.

As the introduction to this chapter has described, the GUI for this framework is implemented under Microsoft C# .NET using Visual Studio 2008. Visual Studio 2008 offers a drag-and-drop environment for building a GUI, thus all code is automatically generated when the GUI is built. Therefore, it does not prove viable to provide the code behind the GUI since it is all arbitrary.

4.3 Shell Process Implementation

In order to run Bash commands under the C# programming language, a process must be created to invoke a Bash shell. The code used is a modified version of the ProcessCaller originally provided by Mayer (2003).

```
private ProcessCaller processCaller;

public void runBashCommand(string arguments)
{
      processCaller = new ProcessCaller(this);
      processCaller.FileName = "bash";
      processCaller.Arguments = arguments;
      processCaller.StdErrReceived += new
      DataReceivedHandler(writeStreamInfo);
      processCaller.StdOutReceived += new
      DataReceivedHandler(writeStreamInfo);
      processCaller.Completed += new
      EventHandler(processCompletedOrCanceled);
      processCaller.Cancelled += new
      EventHandler(processCompletedOrCanceled);
      this.status.Text = Environment.NewLine;
      processCaller.Start();

}
```

This code will call a new process with the `processCaller.FileName` set to “bash”. Any arguments given will be passed to a Bash shell. All output from Bash will then be redirected to `this.status.Text` which is the Bash Output textbox described in Section 4.2. To give an example, the following code will use the process caller and print “hello world” to the Bash Output textbox. The `-c` flag is used to read a string input as a command.

```
this.runBashCommand("-c ' "+"echo hello world"+" ' ");
```

4.4 Attack Traffic Component Implementation

In the attack traffic component, four subcategories which are separated by tabs exist. These subsections are labelled Surveillance/Probing, User Privilege Gain, Malicious Software and Denial of Service. As Section 3.3 demonstrated, the purpose of each of these subsections is to invoke a specific category of live attack. The tools used for the attack traffic component implementation include Nmap, Hydra and Hping3.

Within each subsection, a checkbox with the text “Enabled” is provided. If this checkbox is checked, then the attack is considered active and will be invoked once the “Invoke Attack(s)” button is pressed, otherwise, the attack is not carried out.

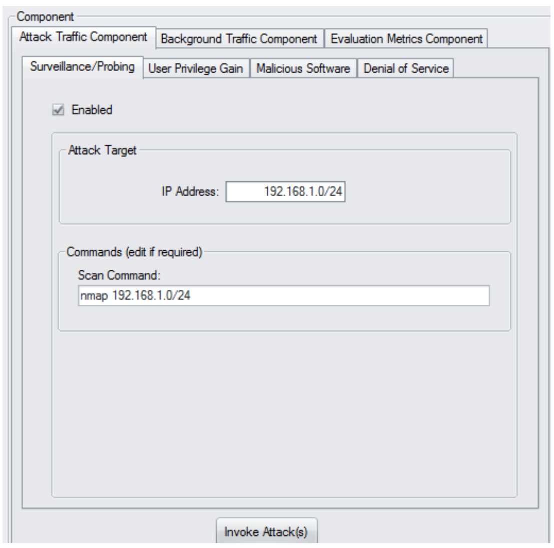
```
private void runAttackBtn_Click(object sender, EventArgs e)
{
      if (surprobChkBox.Checked == true)
      {
            ...
            this.runBashCommand("-c ' " + nmap_ping + " ' ");
            this.runBashCommand("-c ' " + nmap_syn + " ' ");
            this.runBashCommand("-c ' " + nmap_udp + " ' ");
      }
      if (usrGainChkBox.Checked == true)
      {
      ...
      this.runBashCommand("-c ' " + hydraTelnetCommand + " ' ");
      this.runBashCommand("-c ' " + hydraFTPCommand + " ' ");
      }
      if (dosChkBox.Checked == true)
      {
            ...
            this.runBashCommand("-c ' " + Hping3_ping + " ' ");
            this.runBashCommand("-c ' " + Hping3_SYN + " ' ");
            this.runBashCommand("-c ' " + Hping3_LAND + " ' ");
      }
      if (malsoftwareChkBox.Checked == true)
      {
            ...
            this.runBashCommand("-c ' " + qaz + " ' ");
            this.runBashCommand("-c ' " + sober + " ' ");
            this.runBashCommand("-c ' " + deepthroat + " ' ");
            this.runBashCommand("-c ' " + waledac + " ' ");

      }
```

4.4.1 Surveillance/Probing Implementation

The Surveillance/Probing subcategory is implemented using Nmap. Users are able to carry out a default port scan. Upon enabling the attack, entering the IP address of the attack target the command textboxes will automatically update itself to reflect the bash command which will be carried out upon pressing the “Invoke Attack(s)” button. The code implemented is shown as follows:

```
private void nmapTarget_TextChanged(object sender, EventArgs e)
        {
         ...
         nmapScanCommandBox.Text = "nmap " + target;
        }
```

Optionally, if the command presented in the command box is not desirable this can be simply edited. The `target` is a string variable, and will contain an IP address. An example of Nmap configured to carry out a scan on the network 192.168.1.0/24 is shown in Figure 4.4.1.



Enabled: ✓

Attack Target

IP Address: 192.168.1.0/24

Commands (edit if required)

Scan Command:

```
nmap 192.168.1.0/24
```

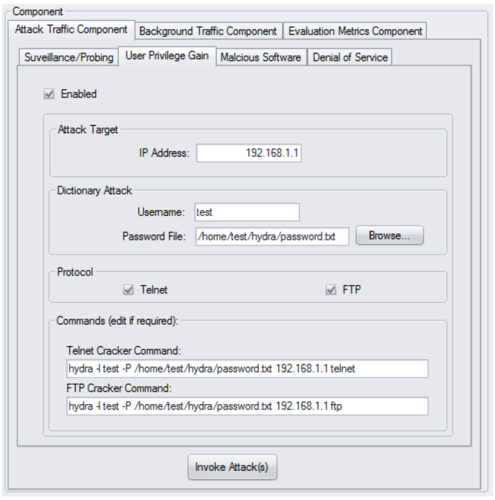
Figure 4.4.1 – Example Configuration of Surveillance/Probing

The string presented in each of the Command boxes will be read and parsed to a bash shell once the “Invoke Attack(s)” button is pressed. If a certain scan is not checked, then its’ command box will default to empty hence no attack will be ran in that type of attack. This idea is applied for each of the other three attacks also.

4.4.2 User Privilege Gain Implementation

The tool Hydra is used in order to create user privilege gain attacks. Hydra allows for a dictionary brute force attack on multiple protocols, including Telnet, HTTP, FTP, SSH as examples. This application has implemented Telnet and FTP brute force attacks only. Users are free to edit the command box to invoke attacks on other protocols.

An example configuration of Hydra which invokes a dictionary attack against target 192.168.1.1 using the password file “password.txt” and be carried out on both the Telnet and FTP protocol is shown (Figure 4.4.2). It should be noted that the password.txt file must be provided manually. This should contain a list of all phrases which Hydra should attempt to use as a password against the target.



Enabled: ✓

Attack Target

IP Address: 192.168.1.1

Dictionary Attack

Username: test

Password File: /home/test/hydra/password.txt

Protocol

Telnet: ✓ FTP: ✓

Commands (edit if required):

Telnet Cracker Command:

```
hydra -l test -P /home/test/hydra/password.txt 192.168.1.1 telnet
```

FTP Cracker Command:

```
hydra -l test -P /home/test/hydra/password.txt 192.168.1.1 ftp
```

Figure 4.4.2 – Example Configuration of User Privilege Gain Attacks

The target IP address, username and a text file containing a list of passwords is required to carry out a brute force dictionary attack. Upon selecting which protocols the attack should be carried out against the Command Box will once again automatically update itself to reflect the bash command which will be used.

```
private void hydraTelnet_CheckedChanged(object sender, EventArgs e)
{
      ...
      if (hydraTelnet.Checked == true)
         {
         this.hydraTelnetCommandBox.Text = "hydra -l " + username
         + " -P " + passwordList + " " + target + " telnet";
         }
      else
      this.hydraTelnetCommandBox.Text = null;
}

private void hydraFTP_CheckedChanged(object sender, EventArgs e)
{
      ...
      if (hydraFTP.Checked == true)
         {
         this.hydraFTPCommandBox.Text = "hydra -l " + username + " -P
         " + passwordList + " " + target + " ftp";
         }
      else
      this.hydraFTPCommandBox.Text = null;

}
```

4.4.3 Malicious Software Implementation

Malicious software consists of worms, viruses and Trojan horses. The issue with implementing this category of attack is that there is a great security risk if actual malicious software was used in testing the IDS. Therefore, instead of using real worms or viruses, this category of attack is achieved by packet crafting through Hping3.

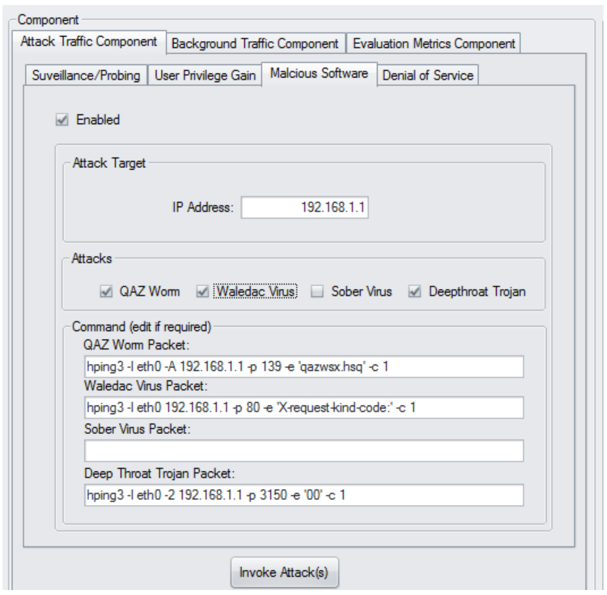
Malicious software can be identified via their propagation signatures (Buchannan, 2009, p. 50) therefore, packets can be crafted using Hping3 that carry out, and contain, the same patterns that a malicious software would have. Four types of malicious software attacks are implemented for this application. This includes Qaz Worm, Waledac Virus, Sober Virus and Deepthroat Trojan. The code used to craft such software, via Hping3, is shown as follows:

```
private void qaz_CheckedChanged(object sender, EventArgs e)
{
        ...
        if (qaz.Checked == true)
        {
        this.qazCommandBox.Text = "Hping33 -I eth0 -A " + target + "
        -p 139 -e 'qazwsx.hsq' -c 1";
        }
        else
          this.qazCommandBox.Text = null;
}
```

```
private void waledac_CheckedChanged(object sender, EventArgs e)
{
         ...
         if (waledac.Checked == true)
            {
            this.waledacCommandBox.Text = "Hping33 -I eth0 " + target
            + " -p 80 -e 'X-request-kind-code:' -c 1";
            }
            else
              this.waledacCommandBox.Text = null;
}

private void sober_CheckedChanged(object sender, EventArgs e)
{
          ...
          if (sober.Checked == true)
             {
             this.soberCommandBox.Text = "Hping33 -I eth0 -S "
              + target + "-p 37 -e '.exe' -c 1";
             }
             else
               this.soberCommandBox.Text = null;
        }

private void deepthroat_CheckedChanged(object sender, EventArgs e)
{
        ...
        if (deepthroat.Checked == true)
           {
           this.deepthroatCommandBox.Text = "Hping33 -I eth0 -2
           "+target+ " -p 3150 -e '00' -c 1";
           }
            else
              this.deepthroatCommandBox.Text = null;
        }
```

In the crafting of the Qaz virus, Waledac worm and Deepthroat Trojan, the text which follows the `-e` flag contains the unique signature of that specific attack. Furthermore, the the tcp flag (`-S` for SYN flag) and ports (`-p flag`) which each of these malicious attacks would attempt to connect to is predefined. In the Deepthroat Trojan, the `-2` flag specifies a packet to be sent in UDP mode. Additionally, it should be noted that in each of the predefined attacks, the default output interface used is eth0, specified via the `-I` flag. This is the default interface alias provided in most Linux distributions. Of course, if that does not apply the editing the command box to reflect the machines network interface alias is possible.

To provide for an example, Figure 4.4.3 shows three of four malicious attacks being configured to be run against 192.168.1.1.



[✓] Enabled

Attack Target

IP Address: 192.168.1.1

Attacks

[✓] QAZ Worm [✓] Waledac Virus [ ] Sober Virus [✓] Deepthroat Trojan

Command (edit if required)

QAZ Worm Packet:
```
hping3 -I eth0 -A 192.168.1.1 -p 139 -e 'qazwsx.hsq' -c 1
```
Waledac Virus Packet:
```
hping3 -I eth0 192.168.1.1 -p 80 -e 'X-request-kind-code:' -c 1
```
Sober Virus Packet:

Deep Throat Trojan Packet:
```
hping3 -I eth0 -2 192.168.1.1 -p 3150 -e '00' -c 1
```

Figure 4.4.3 – Example configuration of Malicious Software Attack

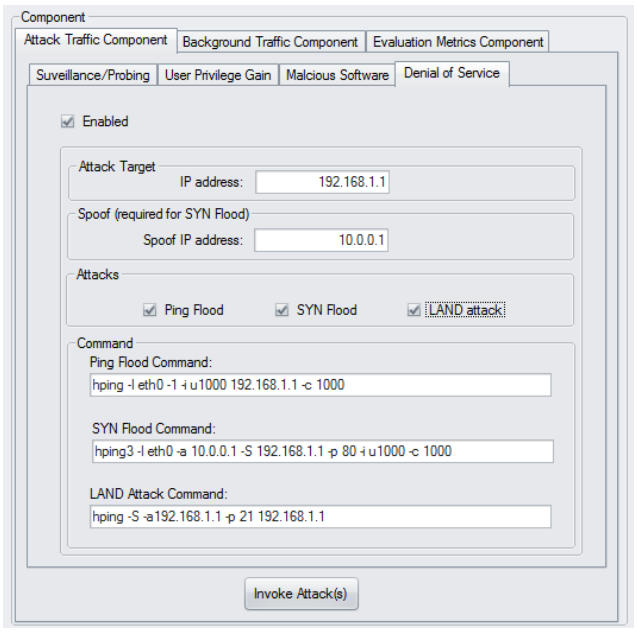
4.4.4 Denial of Service Implementation

Hping3 is once again used for the implementation of Denial of Service attacks. Three main kinds of DoS attacks implemented include SYN flood, PING flood and LAND attack. As with the previous attack implementations, once a user selects the attack target (and a spoofed IP address in the case of a SYN flood using the `-a` flag) and the type of DoS attack to invoke, the Command box will automatically update itself to reflect the Bash command which will be ran.

```
private void synflood_CheckedChanged(object sender, EventArgs e)
{
      ...
      if (synflood.Checked == true)
         {
         this.Hping3CommandBox.Text = "Hping33 -I eth0 -a " + spoof +
         " -S " + target + " -p 22 -i u1000 -c 1000";
         }
         else
           this.hydraFTPCommandBox.Text = null;
}

private void pingflood_CheckedChanged(object sender, EventArgs e)
{
      ...
      if (pingflood.Checked == true)
         {
         this.Hping3CommandBox.Text = "Hping3 -I eth0 -1 -i u1000 " +
         target + " -c 1000";
         }
         else
           this.Hping3PingCommandBox.Text = null;
}

private void landattack_CheckedChanged(object sender, EventArgs e)
{
      ...
      if (landattack.Checked == true)
         {
         this.Hping3CommandBox.Text = "Hping3 -S -a" + target + " -p
         21 " + target + "c -1";
         }
         else
           this.Hping3LANDCommandBox.Text = null;
}
```

In both SYN and PING floods, the default number of request packets sent is 1000 (`-c 1000`), at a rate of 1000 microseconds (`-i u1000`). Although an infinite amount of packets may be sent, this creates difficulty in evaluating an IDS in a meaningful way, especially in calculating the efficiency and effectiveness metrics defined by Sommers *et al.* (2005).

Figure 4.4.4 provides an example configuration of Hping33. The configuration shown will carry out all three DoS attacks against target 192.168.1.1. A spoofed source IP address is required to avoid TCP reset packets from being sent back to the attacker in the case of the SYN flood.


Component
Attack Traffic Component | Background Traffic Component | Evaluation Metrics Component
Suveillance/Probing | User Privilege Gain | Malcious Software | Denial of Service

[✓] Enabled

Attack Target
IP address: 192.168.1.1

Spoof (required for SYN Flood)
Spoof IP address: 10.0.0.1

Attacks
[✓] Ping Flood [✓] SYN Flood [✓] LAND attack

Command
Ping Flood Command:
```
hping -I eth0 -1 -i u1000 192.168.1.1 -c 1000
```
SYN Flood Command:
```
hping3 -I eth0 -a 10.0.0.1 -S 192.168.1.1 -p 80 -i u1000 -c 1000
```
LAND Attack Command:
```
hping -S -a192.168.1.1 -p 21 192.168.1.1
```

Figure 4.4.4 – Example Configuration for Denial of Service Attack

4.5 Background Traffic Component Implementation

The background traffic component of this framework is implemented using Tcpreplay. Tcpreplay is a suite of tools, and consists of tcprep, tcprewrite, tcpreplay, tcpreplay-edit and tcpbridge (Turner & Bing, 2009). A brief summary of the purpose of each tool is provided in the table which follows (Table 4.5.1). A detailed tutorial on the usage of Tcpreplay can also be found in the work of Corsini (2009) in which some practical examples are given (Corsini, 2009, p. 73 – 79).

Table 4.5.1 – Tcpreplay tool suite

Tool	Description
tcpprep	Parses a pcap file (the data set to be used) and establishes a client/server relationship between each of the packets. A cache file is produced which is required for the other tools in Tcpreplay.
tcprewrite	Rewrites TCP/IP and Layer 2 information on the pcap file provided so that it may be routed correctly through a router or switch.
tcpreplay	Replays the pcap file on the network. The speed of playback may be configured.
tcpreplay-edit	Provides the functionality of tcprewrite and tcpreplay as one single command.
tcpbridge	Allows for the bridging of two network interfaces.

In regards to this prototype framework's implementation, the three tools used include: tcpprep, tcprewrite and tcpreplay. These are subcategorised by tabs and labelled "Preparation", "Rewrite Traffic" and "Traffic Playback" respectively in the application. Figure 4.5.1 provides a representation of the process involved in background traffic playback.

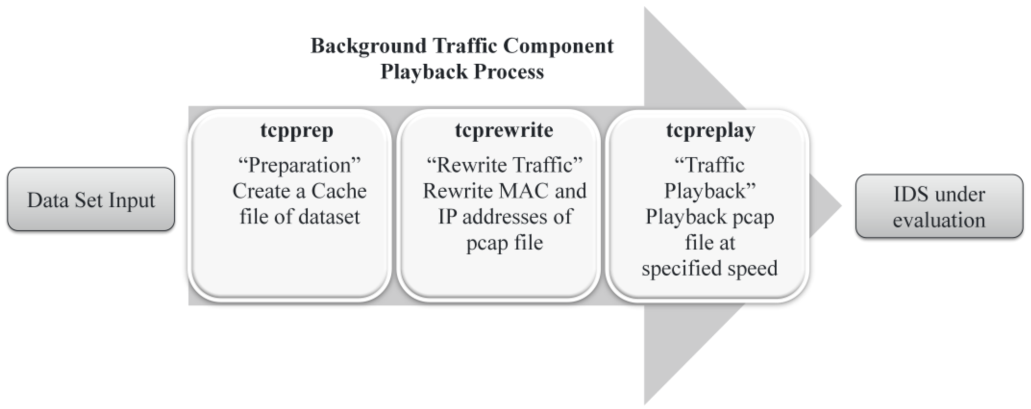

Background Traffic Component Playback Process
Data Set Input
tcpprep
"Preparation" Create a Cache file of dataset
tcprewrite
"Rewrite Traffic" Rewrite MAC and IP addresses of pcap file
tcpreplay
"Traffic Playback" Playback pcap file at specified speed
IDS under evaluation

Figure 4.5.1 – Background Traffic Playback Process

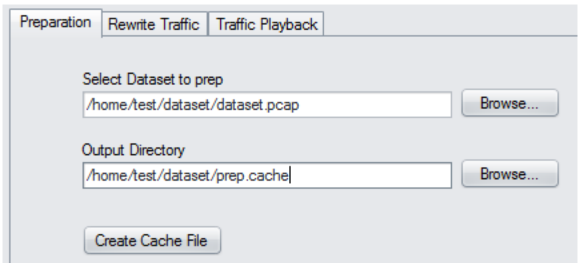
Under the "Preparation" tab, *tcpprep* is used to create a cache file of the pcap file to be played back. The IP and MAC addresses are then rewritten so that the data set may be routed correctly using *tcprewrite* under the "Rewrite Traffic" tab. Finally "Traffic Playback" invokes *tcpreplay* in order to playback the background traffic. This process is represented in. A detailed description on how Tcpreplay was implemented for this application is described next.

4.5.1 Tcprep Implementation

As the previous section demonstrated, the purpose of tcprep is in the prepation of a cache file of the data set which is to be played back. This cache file is relatively small, and simply provides information differentiating between client and server packets. The cache file is required for both tcprewrite and tcpreplay. To create a cache file in this application system, the user specifies the location of the data set to be used and the output directory of the cache file which is to be produced (Figure 4.5.2). Upon clicking the "Create Cache File" button, the following code is invoked:

```
private void createCacheBtn_Click(object sender, EventArgs e)
{
      ...
      this.runBashCommand("-c ' " + "tcpprep " + "-a bridge " + "-i "
      + input_file + " -o " + output_file + " ' ");

}
```

The code shown above uses the `-a bridge` flag, which simply means it will take a input_file (the data set), and divide it into a client server relationship and output this information as a cache file. The output_file is both the directory in which the file is saved to along with the name of the file.



Select Dataset to prep
/home/test/dataset/dataset.pcap

Output Directory
/home/test/dataset/prep.cache

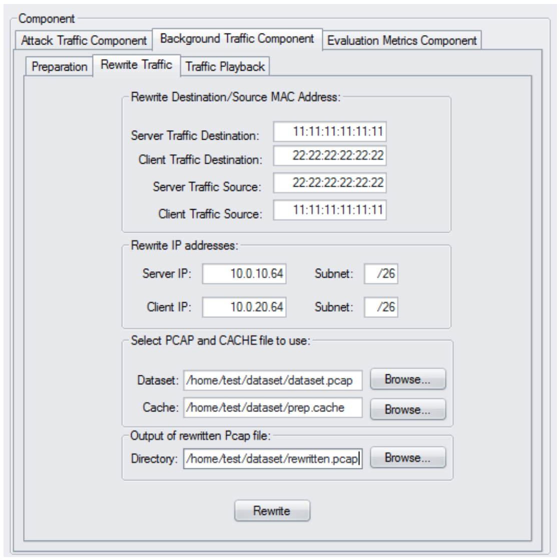
Figure 4.5.2 – Example of tcprep Implementation

4.5.2 Tcprewrite Implementation

The second subcategory within the background traffic playback component is tcprewrite. The purpose of this tool is to parse a data set and rewrite both MAC and IP addresses. This process will produce a rewritten data set so that traffic will flow successfully through devices including routers, firewalls and switches. The cache file (as produced in Section 4.5.1) is also required so that tcprewrite knows which packets are the client and which packets are the server when rewriting the pcap file. The code to rewrite a pcap file is shown as follows:

```
private void rewrite_btn_Click(object sender, EventArgs e)
{
...
this.runBashCommand("-c ' " + "tcprewrite " + "--enet-dmac=" +
dserverMAC + "," + dclientMAC + " --enet-smac=" + sserverMAC + "," +
sclientMAC + " -e " + serverIP + serverSubnet + ":" + clientIP +
clientSubnet + " -C -c " + tcprewritePREP + " -i " + tcprewritePCAP +
" -o " + outputdirectory + " ' ");
}
```

The code shown above takes a input pcap file and rewrites it based on the TCP/IP and Layer 2 information provided (IP and MAC addresses). As an example, Figure 4.5.3 provides a screenshot of the tcprewrite implementation. In the figure provided, the data set known as data set.pcap will be rewritten as rewritten.pcap. The client of this data set will have the source MAC address of 11:11:11:11:11:11 and the server will have the source MAC address of 22:22:22:22:22:22. All clients will be assigned an IP address within the range of 10.0.20.65 - 10.0.20.126 whilst server IP addresses will be within the range of 10.0.10.65 - 10.0.10.126.



Rewrite Destination/Source MAC Address:

Server Traffic Destination: 11:11:11:11:11:11
Client Traffic Destination: 22:22:22:22:22:22
Server Traffic Source: 22:22:22:22:22:22
Client Traffic Source: 11:11:11:11:11:11

Rewrite IP addresses:

Server IP: 10.0.10.64 Subnet: /26
Client IP: 10.0.20.64 Subnet: /26

Select PCAP and CACHE file to use:

Dataset: /home/test/dataset/dataset.pcap Browse...
Cache: /home/test/dataset/prep.cache Browse...

Output of rewritten Pcap file:

Directory: /home/test/dataset/rewritten.pcap Browse...

Rewrite

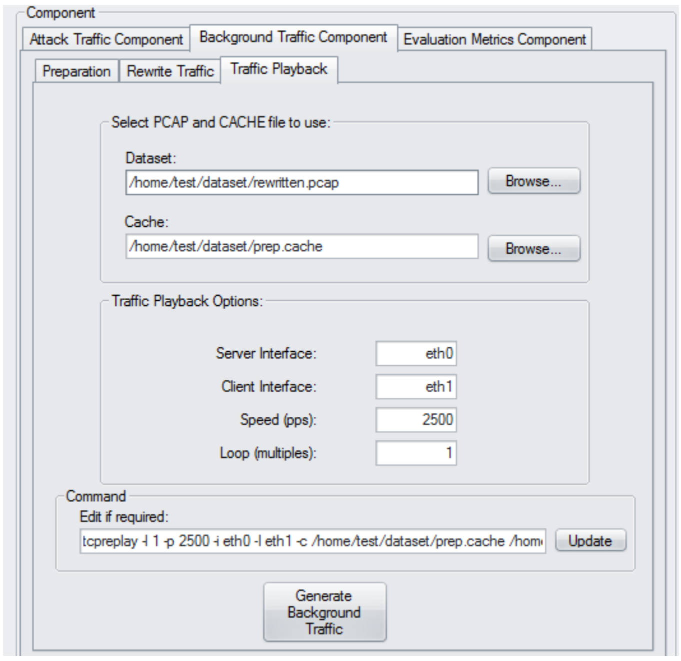
Figure 4.5.3 – Example Configuration of the tcprewrite Implementation

4.5.3 Tcpreplay Implementation

Tcpreplay forms the third and last subcategory of the traffic playback component. It is at this stage that playback of the data set will occur. A user specifies a data set and cache file (created from Sections 4.5.2and Section 4.5.1respectively) along with the output interface for client traffic, and server traffic. Finally, the speed of playback is defined based on packets per second along with have many times the data set should be played back. The code implementing tcpreplay is as follows:

```
private void bgtrafficBtn_Click(object sender, EventArgs e)
{
...
this.runBashCommand("-c ' " + "tcpreplay " + "-l " + loop + " -p " +
speed + " -i " + intCA + " -I " + intUA + " -c " + cacheLocation + "
" + datasetLocation + " ' ");
}
```

As before, an example is given (Figure 4.5.4). In this example, the data set rewritten.pcap will be played back using the cache file prep.cache. All server traffic will be played out of interface eth0 whilst client traffic is played out of eth1. A playback speed of 2500 packets per second is specified. It will only be played back once on the network, as Loop 1 specifies.



Select PCAP and CACHE file to use:

Dataset: /home/test/dataset/rewritten.pcap

Cache: /home/test/dataset/prep.cache

Traffic Playback Options:

Server Interface:	eth0
Client Interface:	eth1
Speed (pps):	2500
Loop (multiples):	1

Command

Edit if required:

```
tcpreplay -l 1 -p 2500 -i eth0 -I eth1 -c /home/test/dataset/prep.cache /hom
```

Figure 4.5.4 – Example of tcpreplay Implementation

4.6 Evaluation Metrics Component Implementation

Similar to the background traffic component, the implementation of the evaluation metrics component of this framework consists of three subcategories separated once again by tabs. They are named “Alarm Retrieval”, “Baseline Results” and “Metric Generation”. Figure 4.6.1 is provided to provide a diagrammatic representation of the processes involved in evaluation metrics component.

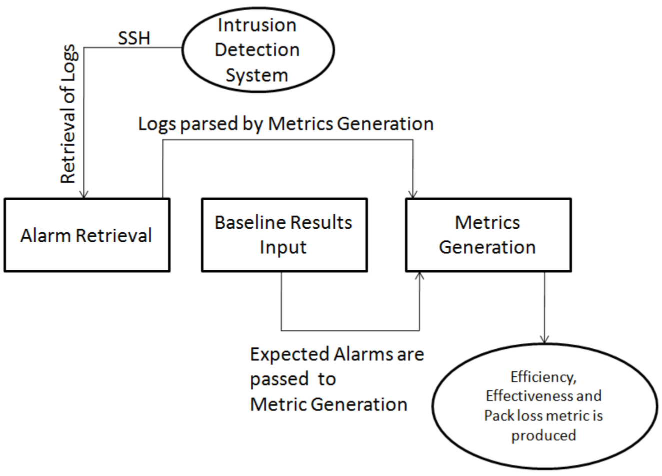

SSH
Intrusion Detection System
Retrieval of Logs
Logs parsed by Metrics Generation
Alarm Retrieval
Baseline Results Input
Metrics Generation
Expected Alarms are passed to Metric Generation
Efficiency, Effectiveness and Pack loss metric is produced

Figure 4.6.1 – Evaluation Metrics Component Diagram

The first subcategory, alarm retrieval, invokes a SSH connection on the system in which the IDS is running on. Alarms and log files are then copied over to the system in which the application resides. The baseline results subcategory is required in order to establish what the expected alarms in each attack should be. The manual input of the expected alarms for each category of attack is required. The metric generation subcategory is then used to analyse the log files retrieved. Based on the baseline results provided, calculation of efficiency and effectiveness metrics is carried out. The metric generation process also returns the result of any packet loss as reported by the IDS.

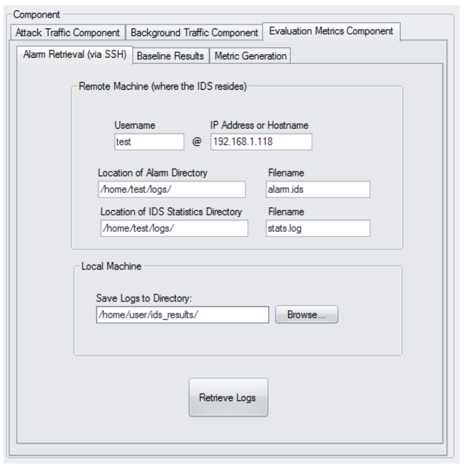
For this application, as mentioned in the introduction of Chapter 3, the intention is to test it against the signature-based IDS Snort. Thus, the evaluation metrics component has been coded to tailor specifically for this IDS. However, with a few modifications to the code, the component can easily be applied to any other IDS. A detailed report on how each of the three subcategories (alarm retrieval, baseline results and report generation) is implemented follows.

4.6.1 Alarm Retrieval Implementation

Alarm retrieval performs the function of copying alarms and log files produced by the IDS after running the attack/background traffic components. The user is required to specify a username and IP address/name of the machine in which the IDS resides on. Furthermore, the location (directory path and name of file) of the alarms and logs file is required. Upon clicking the "Retrieve Log" button, a SSH connection is invoked and the log and alarms file is copied over to the machine in which the application is running on. The code carried out is as follows:

```
private void btnRetrieveLogs_Click(object sender, EventArgs e)
{
      this.runBashCommand("-c ' " + "scp " + username + "@" + target
      + ":" + remotedirectoryAlarm + filenameAlarm + " " +
      localdirectory + " ' ");

      this.runBashCommand("-c ' " + "scp " + username + "@" + target
      + ":" + remotedirectoryStatistics + filenameStatistics + " " +
      localdirectory + " ' ");

      this.runBashCommand("-c ' " + "echo Log has been retrieved" + "
      ' ");
}
```

Figure 4.6.2 provides an example of the alarm retrieval implementation. In this example, the application will connect to *test@192.168.1.118* and copy over alarm.ids and stats.log from the directory */home/test/logs/* and save it to */home/user/ids_results/* directory on the local machine.



Remote Machine (where the IDS resides)

Username: test @ IP Address or Hostname: 192.168.1.118

Location of Alarm Directory: /home/test/logs/ Filename: alarm.ids

Location of IDS Statistics Directory: /home/test/logs/ Filename: stats.log

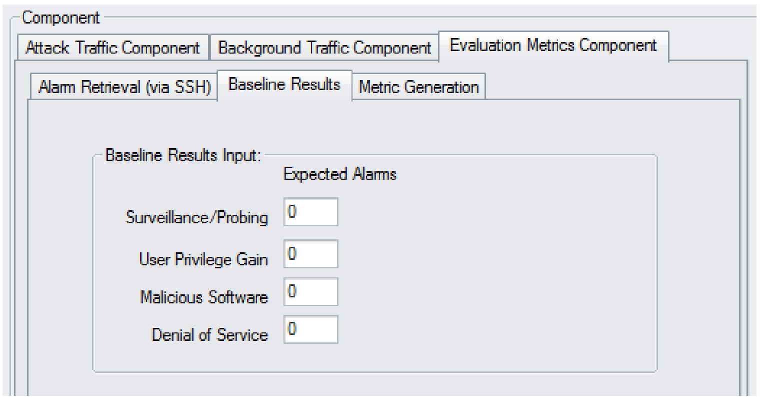
Local Machine

Save Logs to Directory: /home/user/ids_results/

Figure 4.6.2 – Example of Alarm Retrieval Implementation

4.6.2 Baseline Results Implementation

The purpose of the baseline results subcategory is to provide information on expected alarms for the generation of effectiveness and efficiency metrics. Figure 4.6.3 provides a screen shot of what the baseline result subcategory looks like.



	Expected Alarms
Surveillance/Probing	0
User Privilege Gain	0
Malicious Software	0
Denial of Service	0

Figure 4.6.3 – Example of baseline results Implementation

As the figure presented shows, each of the four categories of attacks implemented for this application are present. To establish a baseline, all attacks which are to be evaluated against the IDS must be manually ran against the IDS first – one after the other. During this initial baseline test, no background traffic should exist.

Running a single attack, with no background traffic, should put very little stress on the IDS. It is expected at this point that the IDS will detect the attack successfully. However, since we are performing black-box testing, we don’t know how many alarms the IDS may log for each individual attack; thus we establish a baseline of expected alarms. After inputting the expected alarms for each attack, the information can then be used in metric generation (see next section) to calculate the metrics of efficiency and effectiveness.

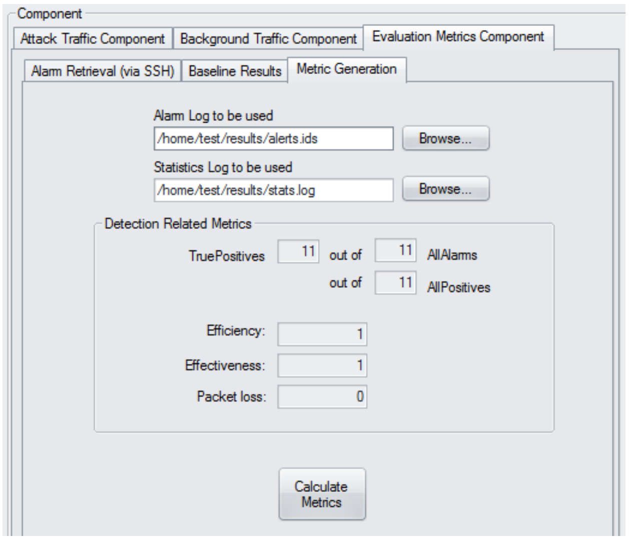
Very little code is required for this part of the implementation, with the exception of converting the input of expected alarms into an integer variable. This is required in order to perform the calculations of metrics in the report generation subcategory since, by default, the input of expected alarms will be read as a string variable. The code used is as follows:

```
int dosBaselineAlarm_int = int.Parse(this.dosBaselineAlarms.Text);

int malsoftwareBaselineAlarms_int =
int.Parse(this.malsoftwareBaselineAlarms.Text);

int usrPrivBaselineAlarms_int =
int.Parse(this.usrPrivBaselineAlarms.Text);

int survBaselineAlarms_int = int.Parse(this.survBaselineAlarms.Text);
```

4.6.3 Metric Generation Implementation

It is within this subcategory of the evaluation component in which the efficiency, effectiveness and packet loss metrics are generated based on the log files produced by the IDS after testing. These log files are known as the alarm log and statistics log, both of which are produced by Snort. As described in Section 4.6.1, the alarm retrieval process will take care of acquiring these logs.

Assuming the files have been retrieved successfully, we simply provide the location of the files. Upon clicking the "Calculate" button, the alarm.ids and statistics.log file is parsed, and based on the information provided efficiency, effectiveness and packet loss metrics are produced automatically as shown in Figure 4.6.4.


Component
Attack Traffic Component
Background Traffic Component
Evaluation Metrics Component
Alarm Retrieval (via SSH)
Baseline Results
Metric Generation
Browse...
Browse...
Detection Related Metrics
Calculate Metrics

Alarm Log to be used: /home/test/results/alerts.ids
Statistics Log to be used: /home/test/results/stats.log
TruePositives 11 out of 11 AllAlarms
out of 11 AllPositives
Efficiency: 1
Effectiveness: 1
Packet loss: 0

Figure 4.6.4 – Metric Generation Example

In this example, the IDS under test logged 11 TruePositives. Since AllPositives (the input from baseline results) and AllAlarms raised were both 11, we have an efficiency and effectiveness of 1. The statistics log provided contained no packet loss, thus the figure shown reflects this.

In regards to the code, three methods are invoked when the "Calculate" button is pressed which include: findAllAlarmsDetected(), findPacketLoss(), and calculateMetrics(). The formulas for calculation of efficiency and effectiveness, as described by Sommers *et al.* (2005), is presented once more for ease of reference when attempting to understand the code which has been written:

$$\text{Efficiency} = \frac{\text{TruePositives}}{\text{AllAlarms}}$$

$$\text{Effectiveness} = \frac{\text{TruePositives}}{\text{AllPositives}}$$

The purpose of the findAllAlarmsDetected() method is to perform a simple regular expression match on the statitics file, in order to find the total alarms raised by the IDS under test (**AllAlarms**). This is achieved using the StreamReader process in C# as the code below shows.

```
public void findAllAlarmsDetected()
{
        ...
        StreamReader streamReader = new StreamReader(statisticslog);
        string fileContent = streamReader.ReadToEnd();

        Match match = Regex.Match(fileContent, @"ALERTS:  (\d+)");

        if (match.Success)
                {
                string capture = match.Groups[1].Captures[0].Value;
                allAlarms.Text = capture;
                }
}
```

The findPocketLoss() method is very similar, except, instead of matching characters after “ALERTS” we perform a match against characters after “DROPPED” which will be a numeric value of packets lost as reported by the IDS.

The calculateMetrics() method is used to perform the calculations required for efficiency and effectiveness metrics. Similar to the previous two methods, the StreamReader process is used, this time to parse the alarms file. A count of alarms raised by each attack is carried out and stored as individual variables using a regular expression. This is achieved since each time Snort logs an alert, the alert will contain a message describing what the alert logged was. We use a regular expression to count how many times certain “messages” show up in the alerts file, hence can calculate how many times Snort logged each attack.

```
string alarmfile = alarmLog.Text;
StreamReader sr = new StreamReader(alarmfile);
string text = sr.ReadToEnd();
sr.Close();

Regex r1 = new Regex(@"\b" + "Portscan" + @"\b",
RegexOptions.IgnoreCase);
MatchCollection mc1 = r1.Matches(text);

..

Regex r10 = new Regex(@"\b" + "LAND Attack Detected" + @"\b",
RegexOptions.IgnoreCase);
MatchCollection mc10 = r10.Matches(text);

double nmap_alarm_count=mc1.Count;
..
double Hping3Land_alarm_count=mc10.Count;
```

The total amount of detected attacks (**TruePositives**) is then calculated by simply adding up the total attacks logged by the IDS. The total number of expected attacks (**AllPositives**) is calculated in the same way by adding up all expected alarms based on the information provided in the Baseline Results subcategory. Finally, the calculation of efficiency and effectiveness metric will occur. The code which carried out this task is as follows:

```
double effectivness = total_detected_alarms / total_expected_alarms;
effectivenessDouble.Text = effectivness.ToString();
```

```
double efficiency = total_detected_alarms / all_alarms_count;
efficiencyDouble.Text = efficiency.ToString();
```

4.7 Conclusion

Using Microsoft C# along with Mono, each of the three components of the framework has been successfully produced. A list of all programs used in this implementation along with their purpose is provided in Table 4.7.1.

The aim of the project is to produce a prototype framework which is capable of evaluation of an NIDS. Having shown the implementation in this chapter, this aim is now partially met since a framework has been successfully produced. However, in order to fulfil the project aim completely, an evaluation of an NIDS, using this implemented framework must take place to show its' capabilities. The next chapter provides details of this evaluation.

Table 4.7.1 – Programs Used In Implementation

Program	**Version**	**Purpose**
Microsoft C# .NET Visual Studio 2008	3.5	Primary IDE used for implementing the actual application.
Monodevelop/Mono	1.0	Secondary IDE used to troubleshoot and run the application from a Linux environment.
Nmap	4.62	Implementation of Surveillance/Probing Attacks.
Hydra	5.4	Implementation of User Privilege Attacks.
Hping3	3.0.0-alpha-2	Implementation of DoS and Malicious Software Attacks.
Tcpreplay	3.3.1 (build 2033)	Implementation for Background Traffic generation.

5 Evaluation

5.1 Introduction

This chapter will outline the evaluation carried out on the NIDS Snort using the framework which has been implemented. The description for this set of experiment, along with test bed and configuration parameters, is provided in Section 5.2. The results and conclusion to the experiment is discussed in Section 5.3 and Section 5.4. A conclusion is provided in Section 5.5.

5.2 Experiment Description

The Attack Traffic Component of the framework is capable of carrying out surveillance/probing, user privilege gain, malicious software and DoS attacks. In each test scenario of this experiment, each and every single attack that the framework is capable of carrying out will be invoked against a target machine. Background traffic will run in conjunction with the attacks using the Background Traffic Component. The Monday Week 1 1998 DARPA data set (M. L. Laboratory, 1998) is used in the playback of background traffic.

As stated by Peisert and Bishop (2007), an experiment which is considered scientifically correct will only ever contain one variation (Peisert and Bishop, 2007, p. 142). In this case, the only variation we apply to each instance of running the experiment will be the playback speed of background traffic. By carrying out the experiment in this way, we can assess whether the volume of traffic on the network will have any detrimental effects to Snort's detection abilities. Thus, a dynamic evaluation of the NIDS is achieved. Figure 5.2.1 below shows a schematic of how the experiment will take place.

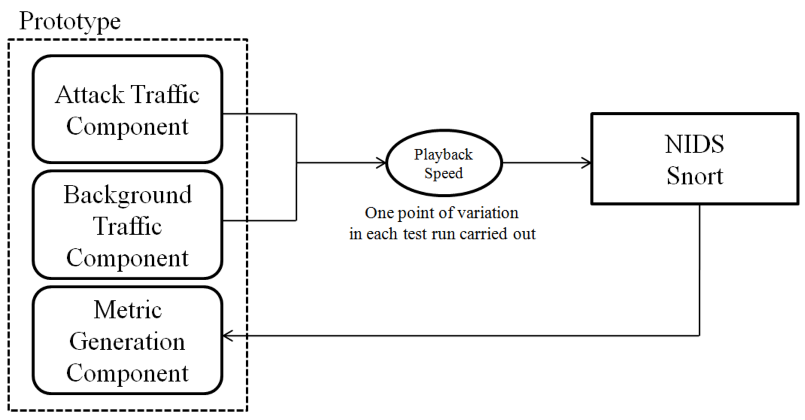

Prototype
Attack Traffic Component
Background Traffic Component
Metric Generation Component
Playback Speed
One point of variation in each test run carried out
NIDS Snort

Figure 5.2.1 – Schematic of Experiment

As described in the Chapter 4, the NIDS known as Snort will be used for this evaluation. Since Snort is a signature-based IDS, rules are required for the detection of attacks. The VRT certified rules from Sourcefire (2009) are used. Furthermore, individual rules are crafted manually to detect each attack which the application is capable of (see Section 5.2.2). This allows us to monitor both the detection metrics of efficiency and effectiveness which is part of the Evaluation Metrics Component of the framework.

A baseline of expected alarms was first established in order to allow for metric generation later on. The experiment was then run a total of six times. In each test, a variation of the playback speed of background traffic was applied whilst every other element remained the same. The six variations of playback speed were: 20Mbps, 40Mbps, 60Mbps 80 Mbps, 100 Mbps and 120Mbps. A limit of 120Mbps playback speed was reached as it was discovered during testing that this was as fast as Tcpreplay was reliably capable of sending out traffic on the virtual environment (see Section 5.2.1 for test bed description). The play speed which were logged is as reported by Tcpreplay.

In each instance of a test, Snort was first started, before background and attack traffic was invoked on the network using the framework. To monitor the memory usage and CPU utilization of the machine in which Snort resides, the *top* command in Linux was used. This command was run in batch mode, which allows the information to be written to a file. The `-b` flag tells *top* to run in batch mode, whilst `-p` is the process ID to monitor (which in this case should be the process ID of Snort). An example of this command is shown as follows:

```
$ top -b -p 5118 >> resourceUtil_metrics
```

After all traffic is sent through the network, Snort is stopped and retrieval of metrics takes place using the Evaluation Metrics component of the framework. These results are logged before a new instance of the test is carried out with a different playback speed.

5.2.1 Test Bed Description

Virtual machines are used to create a private virtual network in order to conduct the experiment. The software VMware (version 6.0.2) (VMWare, 2009) has been used for this purpose. Three virtual machines are required: one machine to run the application, one machine to run Snort and one machine to act as the target of attacks. All three machines are running the Xubuntu Distribution of Linux using kernel 2.6.27. The three machines are connected to a virtual switch which is created from VMWare. The specifications for the three machines are presented in Table 5.2.1. Figure 5.2.2 presents a high level diagram of the virtual network.

Table 5.2.1 – Specifications of Virtual Machines

Machine Name	Operating System	CPU (shared)	Memory
VM 1	Xubuntu Kernel: 2.6.27	Intel Core2 Quad Q6600 @ 2.40 GHz	512 MB
VM 2	Xubuntu Kernel: 2.6.27	Intel Core2 Quad Q6600 @ 2.40 GHz	512 MB
VM 3	Xubuntu Kernel: 2.6.27	Intel Core2 Quad Q6600 @ 2.40 GHz	256 MB

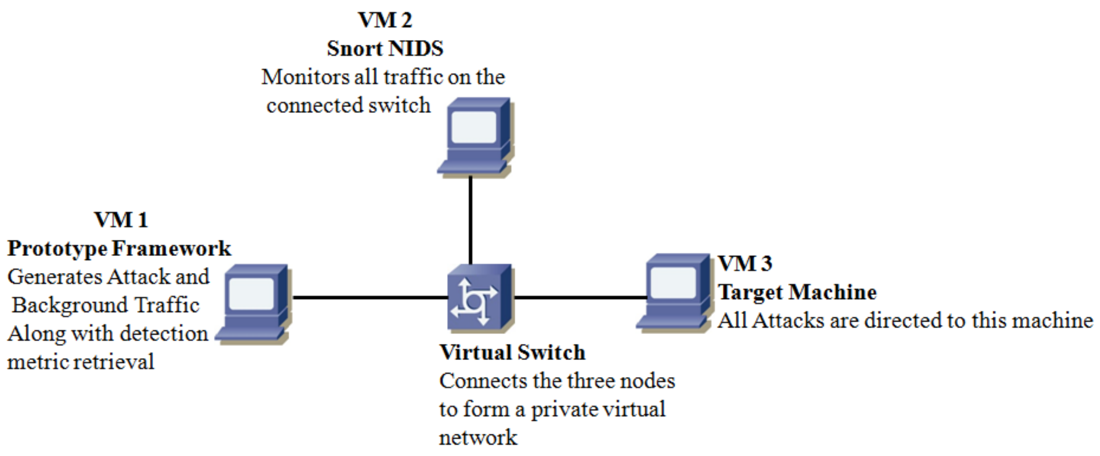

VM 2
Snort NIDS
Monitors all traffic on the connected switch
VM 1
Prototype Framework
Generates Attack and Background Traffic Along with detection metric retrieval
Virtual Switch
Connects the three nodes to form a private virtual network
VM 3
Target Machine
All Attacks are directed to this machine

Figure 5.2.2 – Diagram of Virtual Network Test Bed

As the figure above shows, the framework resides in VM 1. This virtual machine will generate background traffic, along with sending all attacks to VM 3, whilst VM 2 is configured to run Snort and monitor all network traffic seen in this virtual environment. VM 3, the target machine, is configured to accept both Telnet and FTP connections in order to realistically allow for VM 1 to generate attacks on it. After each test is ran with a variation in background traffic playback speed, VM 1 will retrieve the logs produced by Snort and parse them automatically using the Evaluation Metrics Component.

In carrying out the experiments through a virtual environment, the ability to ensure no unexpected traffic will be seen on the network achieved. Furthermore, a greater control over experiments is possible since the three machines form a private virtual network. However, it is acknowledged that one major limitation in carrying out the experiments in this environment is that it may not accurately reflect a real-life network. Unfortunately, due to time limitations, testing using real-life equipment in a laboratory setting was unable to be achieved.

5.2.2 Snort Configuration

Snort Version 2.7.0 (Build 35) is used for this evaluation. As described previously, along with the VRT rule set provided by Sourcefire, some custom rules need to be implemented to detect each of the attacks which the application carries out. The following is a comprehensive list of all custom rules which were created to detect each attack the application is capable of:

```
# Raise alert if more than 3 connections are made to Telnet in 1
second
# seconds.
alert tcp $HOME_NET 23 -> any any  (msg:" Telnet Bruteforce Attack
Detected"; flow:  from_server,established; content:"Password";
nocase; threshold: type threshold, track by_src, count 3, seconds 1;
sid:001;)

# Raise alert if more than 3 connections are made to FTP in 1 second
# seconds.
```

```
alert tcp any any -> $HOME_NET 21 (msg:"FTP Bruteforce Attack
Detected"; flow:  to_server,established; content:"PASS";  threshold:
type threshold, track by_src, count 3, seconds 1; sid:002;)

# Raise alert if packet contains ACK flag, destination port is 139
# and contains content: "|71 61 7a 77 73 78 2e 68 73 71|"
alert tcp any any -> any 139 (msg:"QAZ Worm Detected"; flags:A;
content:"|71 61 7a 77 73 78 2e 68 73 71|"; sid:003;)

# Raise alert if packet destination port is 80 and contains the
# content: "X-Request-Kind-Code:" (note:|3A| is hex)
alert tcp $HOME_NET any -> any 80 (msg:"Waledac Virus Detected";
flow:to_server; content:"X-Request-Kind-Code|3A|"; nocase;
reference:url,blogs.technet.com/mmpc/archive/2009/04/14/wheres-
waledac.aspx; sid:004;)

# Raise alert if a SYN flag is sent to  port 37
alert tcp any any -> any 37 (msg:"Sober Virus Detected";
flow:stateless; flags:S,12; content:".exe"; threshold:type limit,
track by_src, count 1, seconds 60; sid:005;)

# Raise alert if a UDP packet attempts connection to port 3150 with
# content "00".
alert udp any any -> $HOME_NET 3150 (msg:"Deepthroat Trojan
Detected"; flow:to_server; content:"00"; depth:2; metadata:policy
security-ips drop; reference:mcafee,98574; reference:nessus,10053;
classtype:trojan-activity; sid:006;)

# Raise alert if more than 5 ICMP are sent in 1 second with same
src/dst IP
alert icmp any any -> $HOME_NET any (msg:"Ping Flood Detected"; flow:
stateless; threshold: type threshold, track by_src, count 5, seconds
1; sid:007;)

# Raise alert if more than 5 SYNS are sent in 1 second on port 22
alert tcp any any -> $HOME_NET 22 (msg:"Syn Flood Detected"; flow:
stateless; flags:S,12; threshold: type threshold, track by_src, count
5, seconds 1; sid:008;)

# Raise alert if SYN flag attempts to connect a socket to itself
alert tcp any any -> any 22 (msg:"LAND attack Detected"; flags:S;
sameip; sid:009;)
```

For the detection of surveillance/probing attacks, which are carried out using Nmap, Snort has an in-built pre-processor called sfportscan which will detect any attack in this category, thus, there is not a need to create a custom rule for this type of attack. The configuration for sfportscan is as follows:

```
preprocessor sfportscan: proto  { all } \
                         memcap { 10000000 } \
                         sense_level { low }
```

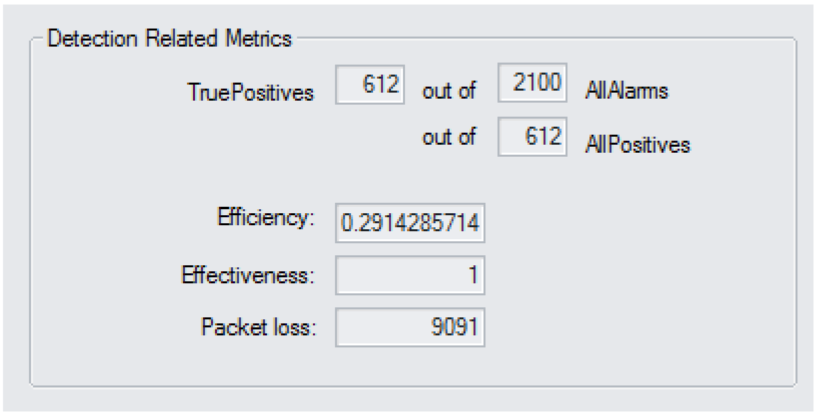
The following command was used to run Snort:

```
snort -i eth0 -l snort_logs/ -c /etc/snort/snort.conf 2> stats.log
```

The configuration file in `/etc/snort/snort.conf` contains Snort's configuration along with a reference to the rules to be used. The `2>` command redirects the output of Snort once it stops running in order to save the statistics Snort reports to a file named `stats.log`. This report contains details of packet loss and all alarms logged which is used parsed by the evaluation metrics component.

5.3 Results

Each test run was carried out by invoking attack and background traffic as described in Section 5.2. After a test run finished running, the Evaluation Metrics component was used to automate the retrieval of logs from the IDS. These files were then automatically parsed by this component and the detection metrics of efficiency, effectiveness and packet loss were reported and made a note of. One example of the Evaluation Metrics Component generating these statistics is presented in Figure 5.3.1 in which the logs retrieved were from the 120Mbps playback.

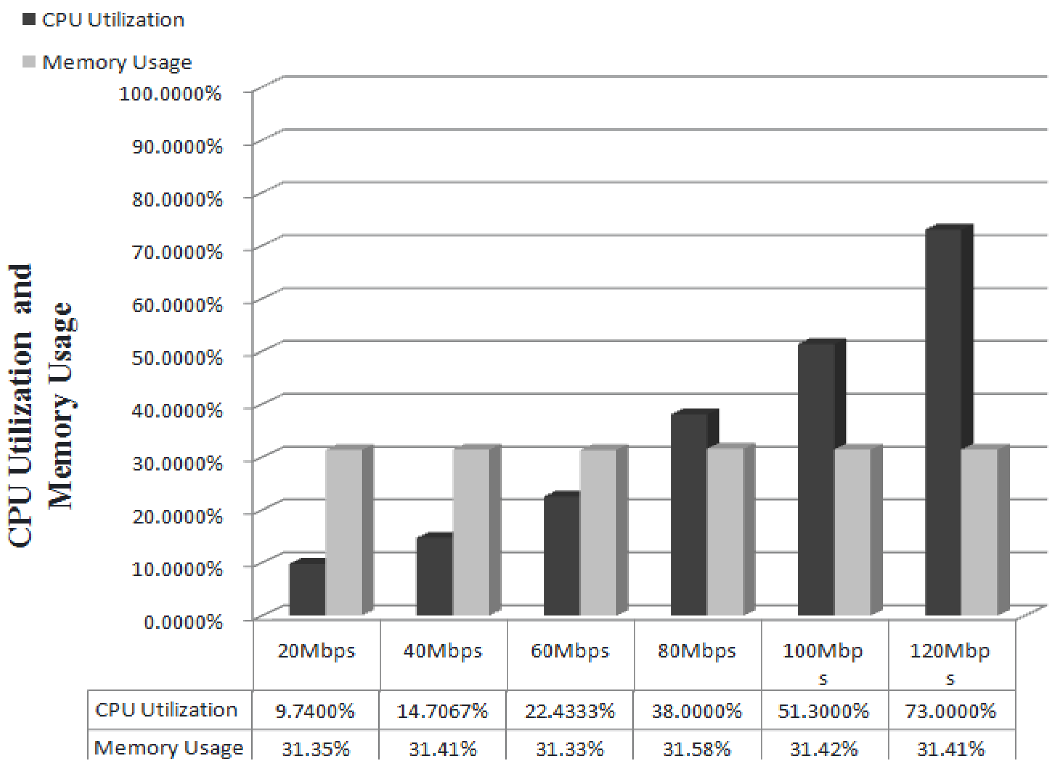

Detection Related Metrics

TruePositives	612	out of	2100	AllAlarms
		out of	612	AllPositives
Efficiency:	0.2914285714			
Effectiveness:	1			
Packet loss:	9091			

Figure 5.3.1 - Detection Metrics Generated for 120Mbps Playback

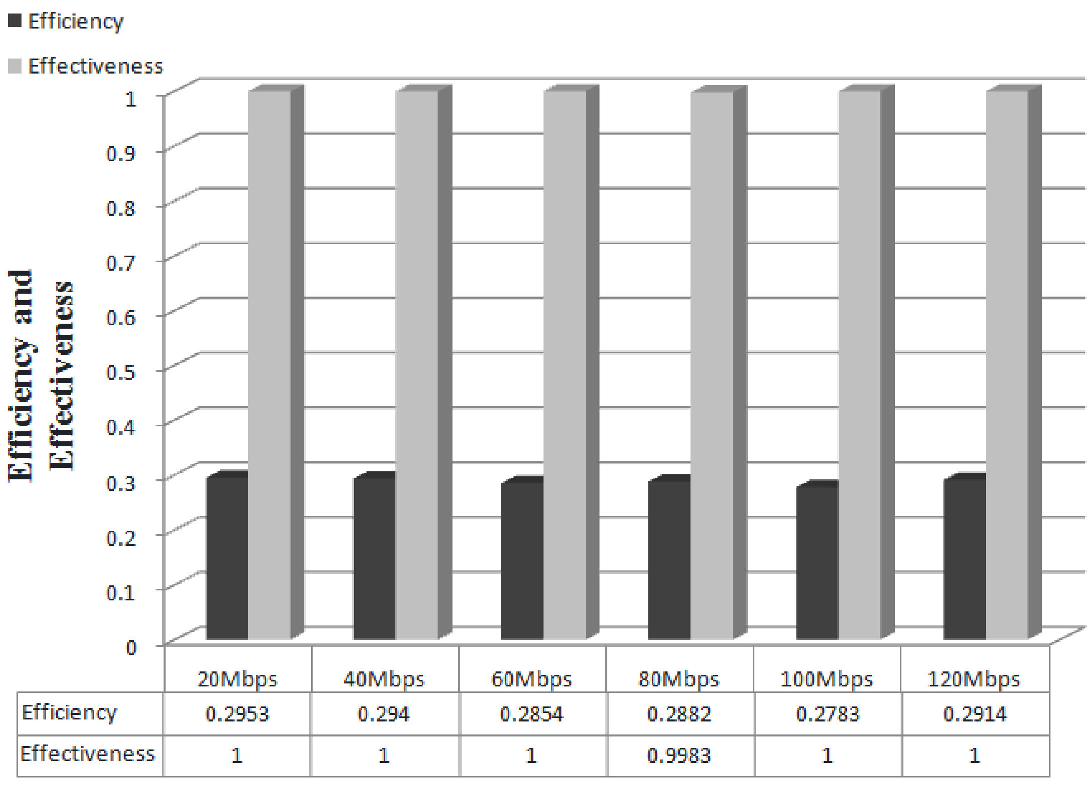
Along with the making a note of the detection metrics, the resource utilisation metrics of CPU Utilisation and Memory usage are also logged. As noted in Section 5.2, the capabilities of the prototype framework does not allow for automated retrieval of resource utilisation metrics thus, these results had to be obtain by manually reading the file which was produced by the *top* command. Having acquired all data required for each of the six tests, the results are compiled into bar charts.

Firstly, Figure 5.3.2 shows CPU utilization and Memory usage in comparison with the different playback speeds. Figure 5.3.3 then shows Efficiency and Effectiveness metrics in comparison with the different playback speeds whilst Figure 5.3.4 shows the packet loss metric. A detailed analysis of the results is provided in the next section.



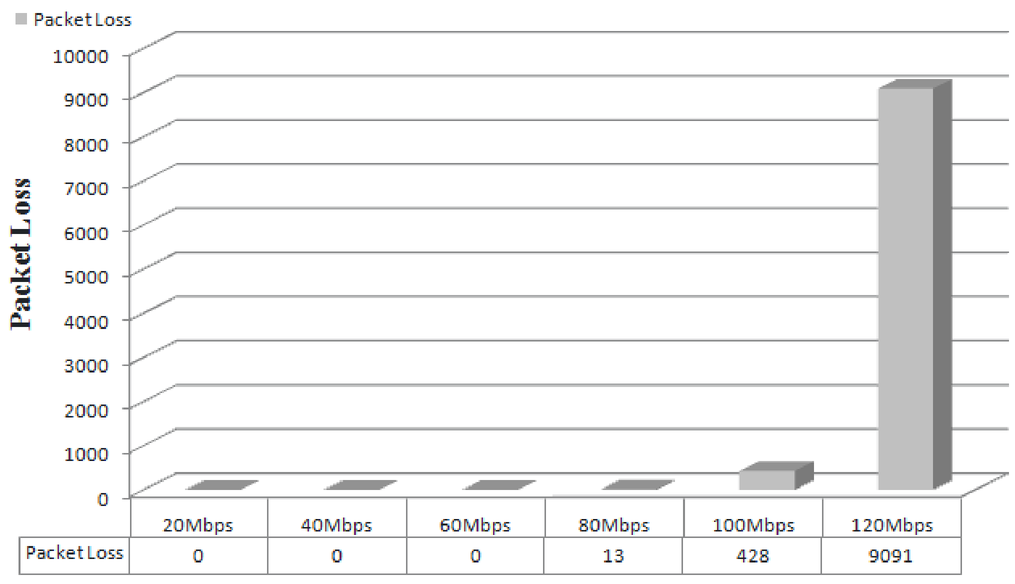
	20Mbps	40Mbps	60Mbps	80Mbps	100Mbps	120Mbps
CPU Utilization	9.7400%	14.7067%	22.4333%	38.0000%	51.3000%	73.0000%
Memory Usage	31.35%	31.41%	31.33%	31.58%	31.42%	31.41%

Figure 5.3.2 –CPU Utilization and Memory Usage Results



	20Mbps	40Mbps	60Mbps	80Mbps	100Mbps	120Mbps
Efficiency	0.2953	0.294	0.2854	0.2882	0.2783	0.2914
Effectiveness	1	1	1	0.9983	1	1

Figure 5.3.3 – Efficiency and Effectiveness Results



	20Mbps	40Mbps	60Mbps	80Mbps	100Mbps	120Mbps
Packet Loss	0	0	0	13	428	9091

Figure 5.3.4 – Packet Loss Metric Result

5.4 Analysis

This evaluation has shown that the framework is capable of generating both attack and background traffic. In regards to background traffic, variable playback speeds was achieved which meant a comparison between the resource utilisation metrics and detection metrics could be achieved. Furthermore, it is demonstrated that the Evaluation Metrics component allows for quick and efficient retrieval of logs produced by the IDS thus results of experimentation could be easily acquired.

The results presented in the previous section shows clearly that there is a direct relation between the increase in traffic and the CPU Utilisation of Snort. At around 40% utilization and above, it appears apparent that Snort will begin to drop packets, as the Packet Loss figure demonstrates. Additionally, there is a relatively high increase in the packets loss once CPU Utilization increases to approximately 70% and above. However, slightly unexpected is that Snort uses the same amount of memory even at the highest playback speed. However, this is not considered a limitation since it shows Snort is quite resourceful in its use of memory even in high volumes of network traffic.

In regards to detection metrics, Figure 5.3.3 shows that Snort is highly effective in the detection of attacks. All but one test instance reported an Effectiveness metric of 1, meaning all attacks were detected successfully. However, the one exception was that during the playback speed of 80Mbps Snort failed to log one attack. In this instance, Snort had reported a packet loss of 13 (Figure 5.4.1). In investigating this result, it was discovered that Snort had dropped an attack packet along with a few arbitrary background traffic packets. Thus, this highlights the absolute importance of an IDS having 100% effectiveness at all times.

However, the results produced in Figure 5.3.3 in regards to the efficiency metric does raise some questions. In each test run, approximately 30% of alerts raised were false-positives. Very little variation of these results were found, and upon running a test in which only background traffic was generated, it was found that the VRT rule set used would raise false-positive alerts due to the background traffic used (which, in this case, was the DARPA 1998 data set). As the purpose of this experiment was to conduct a black-box evaluation using the application implemented, without a greater degree of analysis on both the detection mechanisms used by Snort, individual analysis of the VRT rule set and the behaviour of traffic in the data set, it cannot be said for certain whether it is Snort or the data set used which is at fault at this point in time.

5.5 Conclusion

The results produced from this framework have shown that Snort is highly effective in the detection of attacks. It has also highlighted the fact that Snort's greatest weakness may be due to packet loss. With an exceptional increase in traffic volume, Snort will utilize a greater degree of CPU Utilization. After reaching around 40% and above, packet loss occurs and, in instances such as this experiment has shown, there is the possibility of an attack evading Snort due to the IDS dropping the packet.

Although it has been shown in this Chapter that the prototype framework is capable of providing an evaluation against the NIDS Snort, there is a certain limitation to the experiment carried out which must be acknowledged. The main issue which may provide for unreliable results is the use of a virtual network environment. Although it may be justified that having a virtual network environment provides for control of experimentation, since all three machines share the same computer system, there may also be underlying factors which skewer the results of the evaluation carried out by the framework which are not apparent. Unfortunately, although preferable, a live laboratory for carrying out this evaluation could not be achieved due to time constraints.

6 Conclusion

6.1 Introduction

The main aim of this project was to produce a framework which is capable of carrying out an evaluation of an NIDS. Snort was chosen as the NIDS to evaluate against. The results in the previous chapter have shown that this framework is capable of carrying out an evaluation against Snort therefore meeting the project aim.

This chapter will discuss how the objectives of the project were met (Section 6.2) along with providing a critique of the prototype framework (Section 6.3). A reflection (Section 6.4) on overcoming some of the main difficulties, along with how the project was managed is discussed and, finally, some proposed future work in regards to this project's topic is looked at (Section 6.5).

6.2 Meeting the Objectives

The first chapter outlined three main objectives which were:

1. Review and research the taxonomy of IDSs and existing methodologies for evaluation including the testing methods applied along with metrics of evaluation.

2. Design a framework which can be used to evaluate a NIDS based on the findings of the literature review with justification for design choices made.

3. Implement and evaluate the framework by testing it against an NIDS to see what results are produced in order to assess the capabilities of the framework.

An analysis on how each of the three objectives was met is provided in Section 6.2.1 6.2.2 and 6.2.3 for objectives 1, 2 and 3 respectively.

6.2.1 Objective 1

The first objective was met by providing a comprehensive literature review in the area of IDS for this project. The taxonomy of IDSs was provided covering multiple subjects including detection methods, the common IDS framework and categories of IDSs. Four main methodologies were also reviewed, two offline evaluation methods and two real-time evaluation methods. Additionally, a review was carried out on the types of metrics that are used in IDS evaluation.

From the literature review, it was summarised that current existing methodologies all have their own strengths and weaknesses whilst there is no "correct" metric of measurement only metrics which would provide meaningful results. Therefore, based on the methodologies and evaluation metrics reviewed it was concluded that a solid methodology for evaluation of IDS consisted of three main requirements:

1. Inclusion of Realistic Attack and Background Network Traffic
2. Ease of Automation
3. Inclusion of Meaningful Metrics for Evaluation

Based on this conclusion, a design could then be established in which attempted to meet all three requirements. In other words, by successfully meeting the objective of carrying out research and review on the subject of IDSs, this helped to create three focused issues which needed to be addressed in the design stage of the framework.

6.2.2 Objective 2

The second objective was met by specifying a design of a framework that consisted of three main components: attack traffic component, background traffic component and evaluation metrics component. Furthermore, a GUI was proposed which "wrapped" the three components into a single application which allowed for ease-of-use and automation of functions. In specifying this design, each of the three requirements for a methodology in the evaluation of IDSs - which were concluded in the literature review - would be met.

It was justified in the design that the use of existing programs and tools would allow for the most realistic live attacks possible. The design also stated the use of both detection metrics and resource utilisation metrics in order to provide for meaningful evaluation.

The only limitation to the second objective of this project, designing a framework for evaluation of an NIDS, was attempting to meet the requirement of realistic background traffic as no tool or program is available which allowed for the generation of realistic background traffic. Therefore, it was proposed that Tcpreplay tool, along with the DARPA data set, be used instead. This design choice was justified since it allows for ease-of-repeatability and control of playback speed of traffic. Furthermore, it also means background traffic generation is not limited to specific protocols as long as another data set can be provided.

6.2.3 Objective 3

The third objective was met through implementation of the framework then using it to carry out an evaluation of Snort. Due to having a relatively concise design, the implementation stage was achieved quite smoothly. The main issue was finding a method to invoke the Bash shell from application, but this was resolved with the help of ProcessCaller code provided by Mayer (2003). The results concluded that the framework was capable of carrying out high volume playback of traffic which stresses Snort to the point of dropping packets. This allows for the conclusion that the framework shows that there is a detrimental effect against CPU Utilisation and Packet Loss. Furthermore, it has also shown that even a loss in a very few number of packets may result in Snort missing an attack.

6.3 Critical Analysis

Having shown that both the aim and objectives of this project have been met, this section will provide a critique on the prototype framework. The strengths and weakness of the framework, in comparison with methodologies reviewed in the literature chapter is the key subject of this discussion.

It must first be acknowledged that there are a few limitations with this framework. To most extents, although attack traffic and background traffic was implemented as specified in the design it is felt that realism of traffic playback is still lacking mainly due to the fact that there is no control over aggregation of attack and background traffic. In comparison with works including Rossey *et al.* (2002) and Sommers *et al.* (2006), it may be stated that these methodologies are currently more ideal since they allow for controlled mixtures of benign and attack traffic playback. This is a capability in which the framework does not currently provide. Furthermore, with no other option but the DARPA data set, it must be acknowledged that the background traffic produced may not accurately reflect real life networks.

Another limitation in this framework is the total number of predefined attacks it is capable of generating. To emphasise, the work of Massicotte *et al.* (2006) uses the Metasploit Framework which allows for a huge number of different attacks and exploits which can be also extended through modules. Furthermore, using fragroute enables traffic manipulation techniques which furthers the number of attacks possible. The same type of techniques is used by Sommers *et al.* (2006) in the Trident evaluation. Although the number attacks possible by MACE is only 21 (Sommers *et al.*, 2006, p. 7) this is still significantly higher than what this framework is capable of.

However, at the same time, there are strengths to this framework which are not found in existing methodologies. This strength is found in the Evaluation Metrics component which takes a relatively original approach to carry the generation of detection metrics. Although the metrics defined come from various sources of previous work, there does not appear to have been attempts in the past to simplify this process as has been achieved in this framework since it allows for automated retrieval and generation of detection metrics.

Another strength in this framework is in the implementation of a GUI which wraps all existing tools into one single application. Providing a menu based system allows for ease of use in carrying out an evaluation especially in regards to playing back attack/background traffic. The GUI also provides further automation in that users may select predefined attacks rather than having to manually enter the commands through a Bash shell. The closest comparison to an existing methodology which achieves similar goals is the LARIAT evaluation carried out by Rossey *et al.* (2002). However, this framework differs is that the code is made open and publicly available for anyone to make modifications and extensions to, rather than being restricted to governmental use only.

6.4 Reflection

Upon starting research into the subject area of IDSs, it was discovered that this topic was vastly more complex than initially expected. A great deal confusion was met when attempting to understand articles and papers written about this topic. The greatest obstacle in the research for this project was trying to understand the methodologies employed by researchers in the evaluation of IDSs since these articles and papers were highly technical. However, over time, the more topics that were read the greater the understanding that was developed which allowed for analysation of articles which previously had little meaning. Overcoming of this difficulty also allowed for achieving the first objective in this projective, which was to provide research and review into the subject area of IDSs.

Another area of difficulty faced was the design stage. There was always doubt as to whether the framework design was effective enough for evaluating an NIDS. This issue was overcome by carrying out multiple repetitions in design and testing. To elaborate, a design was first established then tested. Based on the results of testing, refinements were made. Finally, the prototype framework which this project discusses was achieved.

Making emphasis in the design allowed for ease of implementation. However, one issue which required to be overcome was trying to understand and get Tcpreplay work during the implementation stage. The main problem faced was that in the playback of network traffic, only some traffic would be played back (for example, the client side traffic only) rather than the entire data set. It was discovered over time that this was due to errors made in rewriting the data set and after having gained some knowledge on the principles of routing network traffic this issue was overcome.

Project management is the last area of discussion in this section. It is felt that the project has been successfully managed from start to finish. A time plan was created in the form of a gantt chart at the start of this project and attempts were made to meet this schedule. However, it must be acknowledged that there was some deviation in this time plan due to other commitments but attempts were made to ensure work was carried out on the project if it began to fall behind schedule. Furthermore, although the time plan was not met completely as expected, every effort was made to have weekly meetings with the supervisor in regards to the project so that a focused workload could be maintained. The time plan is presented in Appendix 3 whilst the meeting diary sheets are found in Appendix 4.

6.5 Future Work

At this point, more research into the evaluation of IDSs is still essential. Although the framework presented in project has achieved some fundamental rquirements in testing of NIDSs, a great deal of work in this area of research is still both possible and, more importantly, necessary.

Many improvements can be made for future work in regards to the prototype framework. One of the first improvements could be the inclusion of more attack scenarios. In the implementation, only four threat categories exist and the attacks which are possible are miniscule compared to real-life as the critique in Section 6.3 has mentioned. Furthermore, the predefined attacks are hard coded into the source code during this implementation. Although this approach works as the attack may stay be edited via the command box provided, in retrospect, it is felt having separate files which describes each attack (eg. an XML configuration file of each attack) would have been a better solution since it allows users to better customise their own attacks.

A major extension to this framework is the possibility of providing a layered approach to the attack and background traffic generation components. As an example, Fragroute could be employed on top of existing attacks in order to manipulate packets to carry out evasion/insertion attacks. Also of importance would be an aggregation component which allows for control over the mixture of background/attack traffic which is generated. Although the implementation performs aggregation at the network interface level a greater degree of control over the mixture of attack/background traffic is more ideal. An abstract example of this improved framework is presented in Figure 6.5.1.

Another area of work works which is highly important for evaluation of IDSs is an in-depth research into the generation of background network traffic. As the literature review in this project has demonstrated, background traffic generation is still lacking in many areas including both realism and control. It would be an ideal scenario if some form of implementation exists which allows for the control of highly realistic generation of benign traffic. The work of Rossey *et al.* (2002) achieves this goal to some extent but it is not available for public use.

To provide for an example, if an open-source implementation exists which allows a user to select specific traffic "profiles" which caters for all the different types of protocols such as 70% TCP traffic, 20% Telnet traffic and 10% SSH traffic this would allow an evaluation to take place on a test bed which reflects the different environments an IDS was employed on. Furthermore, being open-source would allow it to be modified and improved to cater for any network protocol which is omitted.

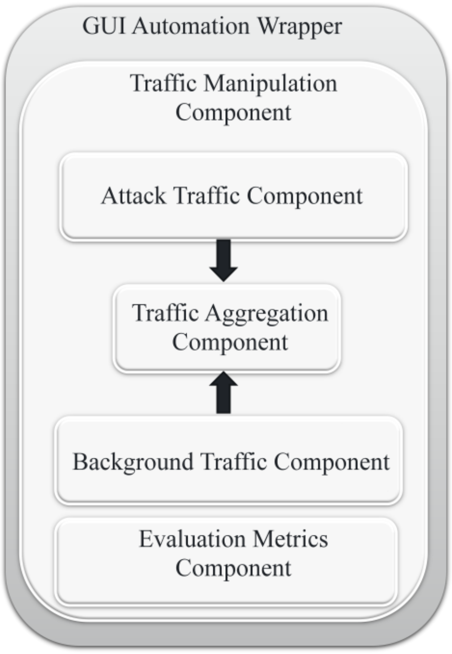

GUI Automation Wrapper
Traffic Manipulation Component
Attack Traffic Component
Traffic Aggregation Component
Background Traffic Component
Evaluation Metrics Component

Figure 6.5.1 – Improvements to Prototype Framework

7 References

Alessandri, D. (2004), Attack-Class-Based Analysis of Intrusion Detection Systems, Master's thesis, University of Newcastle upon Tyne, Newcastle, UK.

Almgren, M., Lundin, E. & Jonsson, B. E. (2003), Consolidation and evaluation of ids taxonomies, *Proceedings of the eighth Nordic Workshop on Secure IT systems NordSec 2003, Gjovik, Norway*, 56-70.

Anzen Computing. (1999), *NIDSBench*, Retrieved 1 November, 2009, from http://packetstormsecurity.nl/UNIX/IDS/nidsbench/nidsbench.html

Athanasiades, N., Abler, R., Levine, J., Owen, H. & Riley, G. (2003), Intrusion detection testing and benchmarking methodologies', *Proceedings of First IEEE International Workshop on Information Assurance*, 2003. IWIAS 2003. 63-72.

Aubert, S. (2002), *HSC – Tools – IDSWakeup*, Retrieved 1 November, 2009, from http://www.hsc.fr/ressources/outils/idswakeup/index.html.en

Axelsson, S. (1999), Research in Intrusion Detection Systems: A Survey, Technical report, *Department of Computer Engineering, Chalmers University of Technology, Sweden*, 1-85.

Beale, J., Baker, A. & Esler, J. (2007), *Snort IDS and IPS Toolkit*, Burlington: Syngress Publishing.

BIS. (2006), Information Security Breaches Survey 2006, Technical report, Department for Business, Innovation and Skills. Retrieved 5 October, 2009, from http://www.berr.gov.uk/files/file28343.pdf

BIS. (2008), Information Security Breaches Survey 2008, Technical report, Department for Business, Innovation and Skills. Retrieved 5 October, 2009, from http://www.berr.gov.uk/files/file45714.pdf

Brugger, S. & Chow, J. (2007), An assessment of the DARPA IDS Evaluation Data set using Snort, Technical report, *Department of Electrical Engineering and Computer Sciences, University of California, Berkeley*.

Buchanan, W. (2009), *Security and Forensics Computing*, Edinburgh Napier University, Edinburgh, UK.

Cavalli, E., Mattasoglio, A. Pinciroli, F. & Spaggiari, P. (2004), Information security concepts and practices: the case of a provincial multi-specialty hospital, *International Journal of Medical Informatics* **73**(3), 297-303.

Corsini, J. (2009), Analysis and Evaluation of Network Intrusion Detection Methods to Uncover Data Theft, Master's thesis, *Edinburgh Napier University, Edinburgh, UK*.

Debar, H., Dacier, M. & Wespi, A. (1999), Towards a taxonomy of intrusion detection systems, *Computer Network* **31**(9), 805-822.

Del Carlo, C., Lakes, S., Illinois, C. & Practical, G. (2003), Intrusion detection evasion: How attackers get past the burglar alarm [White paper], Retrieved 30 March, 2009 from http://www.sans.org/reading_room/whitepapers/detection/intrusion_detection_evasion_how_attackers_get_past_the_burglar_alarm_1284?show=1284.php&cat=detection

Deri, L., Suin, S. & Maselli, G. (2003), Design and Implementation of an Anomaly Detection System: an Empirical Approach, *Proceedings of Terena Networking Conference*.

Fink, G., O'Donoghue, K. F., Chappell, B. L. & Turner, T. G. (2002), A Metrics-Based Approach to Intrusion Detection System Evaluation for Distributed Real-Time Systems, *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IEEE Computer Society, Washington, DC, USA.

Gadelrab, M. S. & El Kalam A. A. (2006), Testing Intrusion Detection Systems: An Engineered Approach, *Proceedings of the 10th IASTED International Conference SOFTWARE ENGINEERING AND APPLICATIONS*, Dallas, TX, USA, 270-275.

Gong, F. (2003), Deciphering Detection Techniques: Part II Anomaly-Based Intrusion Detection [White Paper], *McAfee Security* , McAfee Security White Paper, Retrieved July 8, 2009, from http://www.mcafee.com/us/local_content/white_papers/wp_ddt_anomaly.pdf

Goodin, D. (2009), *Gang Sentenced for UK bank trojan*, Retrieved November 17, 2009 from http://www.theregister.co.uk/2009/11/16/bank_trojan_gang_sentenced/

Graves, J., Buchanan, W., Saliou, L. & Old, J. (2006), Performance Analysis of Network Based Forensic Systems for In-line and Out-of-line Detection and Logging, *Proceedings of the 5th European Conference on i-Warfare and Security (ECIW), Academic Conferences Limited*.

Houle, K., Weaver, G., Long, N. & Thomas, R. (2001), Trends in denial of service attack technology, *CERT Coordination Center* , Technical report, Carnegie Mellon University's Computer Emergency Response Team Coordination Center.

Howard, B., Paridaens, O. & Gamm, B. (2001), Information security: threats and protection mechanisms, *Alcatel Telecommunications Review, 2nd Quarter*, 117-121.

Hping3. (2009), [Computer Software], Retrieved February 15, 2009, from http://www.Hping3.org/

Kargl, F., Maier, J. & Weber, M. (2001), Protecting web servers from distributed denial of service attacks, *WWW '01: Proceedings of the 10th international conference on World Wide Web*, ACM, New York, NY, USA, 514-524.

Kazienko, P. & Dorosz, P. (2003), *Intrusion Detection Systems (IDS) Part I - (network intrusions; attack symptoms; IDS tasks; and IDS architecture)*, Retrieved April 20, 2009 from http://www.windowsecurity.com/articles/Intrusion_Detection_Systems_IDS_Part_I__network_intrusions_attack_symptoms_IDS_tasks_and_IDS_architecture.html.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J. & Das, K. (2000), The 1999 DARPA off-line intrusion detection evaluation, *Computer Networks* **34**(4), 579 - 595.

Lyon, G. (2009) Nmap [Computer Software], Retrieved February 18, 2009, from http://nmap.org/

Massicotte, F., Gagnon, F., Labiche, Y., Briand, L. & Couture, M. (2006), Automatic evaluation of intrusion detection systems. *ACSAC'06: Proceedings of the 22nd Annual Computer Security Applications Conference*, Washington, DC, USA, 361-370.

Mayer, M. (2003), *CodeProject: Launching a process and displaying its standard output. Free source code and programming help*, Retrieved May 2, 2009, from http://www.codeproject.com/KB/threads/launchprocess.aspx?msg=2927207

McHugh, J. (2000), Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, *ACM Transactions on Information and System Security* **3**(4), 262-294.

Mell, P., Hu, V., Lippmann, R., Haines, J. & Zissman, M. (2003), An overview of issues in testing intrusion detection systems, *National Institute of Standards and Technology ITL* , Technical report, NIST IR 7007, Gaithersberg, MD.

Merriam-Webster. (2009). *Merriam-Webster Online Dictionary*, Retrieved September 1, 2009, from http://www.merriam-webster.com/dictionary/security.

Metasploit Project. (2009), *The Metasploit Framework*, Retrieved 1 November, 2009 from http://www.metasploit.com/framework/

M.L Laboratory. (1998), *MIT Lincoln Laboratory: Information Systems Technology*, Retrieved April 8, 2009, from http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1998/training/week1/monday.tar

Mono Project. (2009), *Mono Project* [Computer Software], Retrieved April 29, 2009, from http://www.mono-project.com/Main_Page

Newham, C. & Rosenblatt (1998), *Learning the Bash Shell*, O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Newson, A. (2005), Network threats and vulnerability scanners, *Network Security* **2005**(12), 13-15.

NIST. (2006), *An introduction to computer security: the NIST handbook*, National Institue of Standards and Technology, Ed. U.S Department of Commerce, 2006.
Peisert, S. & Bishop, M. (2007), How to Design Computer Security Experiments, *International Federation for Information Processing Publications- IFIP* **237**, 141 - 148.

Powell, D. & Stroud, R. (2001), MAFTIA: Conceptual Model and Architecture. Project MAFTIA IST-1999-11583 deliverable D2, Technical report, University of Newcastle upon Tyne, Newcastle, UK.

Porras, P., Schnackenberg, D., Staniford-Chen, S., Stillman, M. & Wu, F. (1998), *The common intrusion detection framework architecture*, Retrieved July 31, 2009, from http://gost.isi.edu/cidf/drafts/architecture.txt.

Purcell, J. (2007), Security Control Types and Operational Security [White paper], Retrieved March 29, 2009 from http://www.giac.org/resources/whitepaper/operations/207.php.

Ptacek, T., Newsham, T. & Simpson, H. (1998), Insertion, evasion, and denial of service: Eluding network intrusion detection, Technical report, *Secure Networks, Inc., Calgary Alberta, Canada, Suite 330, 1201 5th Street S.W, T2R-0Y6*.

Quittek, J., Zseby, T., Claise, B. & Zander, S. (2004), Requirements for IP Flow Information Export (IPFIX), Internet Engineering Task Force, Retrieved September 23, 2009 from http://tools.ietf.org/pdf/rfc3917.pdf

Rossey, L. M., Cunningham, R. K., Fried, D. J., Rabek, J. C., Lippmann, R. P., Haines, J. W. & Zissman, M. A. (2001), LARIAT: Lincoln Adaptable Real-time Information Assurance Testbed, Submitted for publication, *IEEE Proc. Aerospace Conference*, p. 2671-2682.

Ranum, M. (2001), Experiences benchmarking intrusion detection systems, NFR Security White Paper, Retrieved April 2, 2009, from http://www.bandwidthco.com/whitepapers/compforensics/ids/Benchmarking%20IDS.pdf

Richardson, L. & Moore, W. (2002), *National Fire Alarm Code Handbook*, Minneapolis, MN: National Fire Protection Association.

Sharma, A., Kumar, R. & Grover, P. S. (2007), A Critical Survey of Reusability Aspects for Component-Based Systems, *Vol. 21, Proceedings of the World Academy of Science, Engineering and Technology*, 35-39.

Singaraju, G., Teo, L. & Zheng, Y. (2004), A Testbed for Quantitative Assessment of Intrusion Detection Systems using Fuzzy Logic, *IWIA '04: Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)*, IEEE Computer Society, Washington, DC, USA.

Snapp, S., Brentano, J., Dias, G., Goan, T., Heberlein, L., Ho, C., Levitt, K., Mukherjee, B., Smaha, S., Grance, T., Teal, D. & Mansur, D. (1991), DIDS (distributed intrusion detection system)-motivation, architecture, and an early application, *Proceedings of the 14th National Computer Security Conference*, 167-176.

Sommers, J. & Barford, P. (2004), Self-configuring network traffic generation, *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, ACM, New York, NY, USA, pp. 68-81.

Sommers, J., Yegneswaran, V. & Barford, P. (2004), A framework for malicious workload generation, *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, ACM New York, NY, USA, 82-87.

Sommers, J., Yegneswaran, V. & Barford, P. (2005), Toward Comprehensive Traffic Generation for Online IDS Evaluation, Technical report, Department of Computer Science, University of Wisconsin, Madison.

Sommers, J., Yegneswaran, V. & Barford, P. (2006), Recent Advances in Network Intrusion Detection Systems Tuning, *CISS '06: Proceedings of the 40th IEEE Conference on Information Sciences and System*, Princeston, NJ, USA, 1490-1495.

Song, D. (2009), Fragroute [Computer Software], Retrieved February 20, 2009, from http://monkey.org/~dugsong/fragroute/

Sourcefire. (2009), Snort [Computer Software], Retrieved February 10, 2009, from http://www.snort.org/

THC. (2009), Hydra [Computer Software], Retrieved September 27, 2009, from http://www.thc.org/thc-hydra/

Tucker, C., Furnell, S., Ghita, B. & Brooke, P. (2007), A new taxonomy for comparing intrusion detection systems, *Internet Research* **17**(1), 88-98.

Turner, A. & Bing, M. (2009), Tcpreplay Tool [Computer Software], Retrieved February 9, 2009, from http://tcpreplay.sourceforge.net/

Vigna, G., Robertson, W. & Balzarotti, D. (2004), Testing network-based intrusion detection signatures using mutant exploits, *Proceedings of the 11th ACM conference on Computer and communications security'*, ACM New York, NY, USA, 21-30.

VMWare. (2009), VMWare [Computer Software], Retrieved February 29, 2009, from http://www.vmware.com/

Wagh, A. (2009), *Purpose of Scanning the Network: Stealth Attacks, Hackers Enigma*, Retrieved 20, September 2009 from http://www.hackersenigma.com/network-security/purpose-of-scanning-the-network-stealth-attacks-2/.

Whitman, M. & Mattord, H. (2008), *Principles of information security*, Canada: Course Technology.

Workman, M. (2007), Gaining Access with Social Engineering: An Empirical Study of the Threat. Information Systems Security, 16(6), 315-331. Retrieved September 6, 2009, from ABI/INFORM Global.

Zhengbing, H., Zhitang, L. & Junqi, W. (2008), A novel Network Intrusion Detection System (NIDS) based on signatures search of data mining, *Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*, ICST, Brussels, Belgium, Belgium

Appendix 1 Initial Project Overview

Title: **Dynamic Performance Evaluation of Intrusion Detection Systems (IDS)**

Overview of Project Content and Milestones

The main evaluation focus in this project will be on signature-based IDS such as "Snort". This project aims to provide an evaluation on the performance of signature-based IDS in relation to the actual true-positive rate under various network conditions and activity through experimentation.

Project milestones are as follows:

- To research the existing technology and the main motivations behind the need for IDS
- To research and understand any existing methods which are used to carry out dynamic evaluation of IDSs.
- Research and design a range of experiments which will prove viable in testing the IDSs in terms of their performance in handling and detecting different types of network traffic activity and threats.
- Implement the experiments by creating or using existing traffic playback tools which will make it possible to carry out such experiments on the Intrusion Detection System
- Undertake research and implement the necessary methods in carrying out performance metric evaluation for actually evaluating how well the IDS performed after testing it with the implemented tools. Such performance metric includes: CPU utilisation, bandwidth usage and memory usage.
- To provide an evaluation of true-positive alarms registered on an IDS based on various different traffic loads, for example: how many true-positives are registered on various types of network traffic based on high traffic load, medium traffic load and low traffic load.
- Reflect on the test results based on expected results and produce overall conclusion.

The Main Deliverable(s)

An implementation of a range of experiments which will test IDSs for true-positive alarms in relation to variable network traffic conditions. Such tests and the evaluation it produces will be integrated as an automated utility.

The results of such experiments and the implementation itself will be detailed in the final report.

The Target Audience for the Deliverable(s)

The target audience will consist of researchers interested in the performance evaluation of IDS along with users who wish to either carry out their own testing on IDS or are interested in the results of such an evaluation. Specifically, such users will include people working or have interest within the security and computer networks field.

The Work to be Undertaken

This work will firstly involve the investigation into the current performance within IDS in general and methods which are used to test them. It will also look at any existing methods used to evaluate an IDS.

Design and implement tools which will allow for the testing of IDSs under various network conditions.

Such an implemented tool should also analyse the performance metric involved and come to a conclusion as to how effective the IDS is in detecting true-positives in relation to the number of intrusions actually sent.

Additional Information / Knowledge Required

Acquire in-depth knowledge of Ds.

Understand performance metrics and how they are evaluated.

Understand and successfully implement tools which will allow for the evaluation of IDS.

Information Sources that Provide a Context for the Project

Journals, articles and papers relating to the topic of IDS and computer networks in general will all be relevant. Some sample papers which have already been looked at for background reading, and prove relevant to this project are as follows:

- Graves, J., Buchanan, W. J., Saliou, L., & Old, J. (2006). Performance Analysis of Network Based Forensic System for In-line and Out-of-line Detection and Logging. *European Conference on Information Warfare and Security.*

- McHugh, J. (2000). Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security , 3* (4), 262–294.

- Paulauskas, N., & Skudutis, J. (2008). Investigation of the Intrusion Detection System "Snort" Performance. *Electronics and Electrical Engineering , 7* (87).

- Sommers, J., Yegneswaran, V., & Barford, P. (2005). Toward Comprehensive Traffic Generation. *UW Technical Report* , 1-12.

Additional sources of information Information from the various websites which relate the tools and software to be used for this project will also provide a good context for learning and understanding.

The Importance of the Project

In terms of computer networking, the whole purpose of an IDS is to detect any traffic which has been deemed dangerous or simply undesired. Such unwanted traffic should, in theory, always be detected by the IDS. It is considered of great importance that an IDS will always raise an alarm whenever unwanted network activity passes through the network since this is the job of the IDS. However, in practice, this is not always the case due to the shortcomings of the IDS.

Thus, the importance of this project is to evaluate such IDS by designing and implementing tests so that we are able to come to a conclusion as to how successful an IDS is when it comes to detecting unwanted network traffic under various conditions.

The Key Challenge(s) to be Overcome

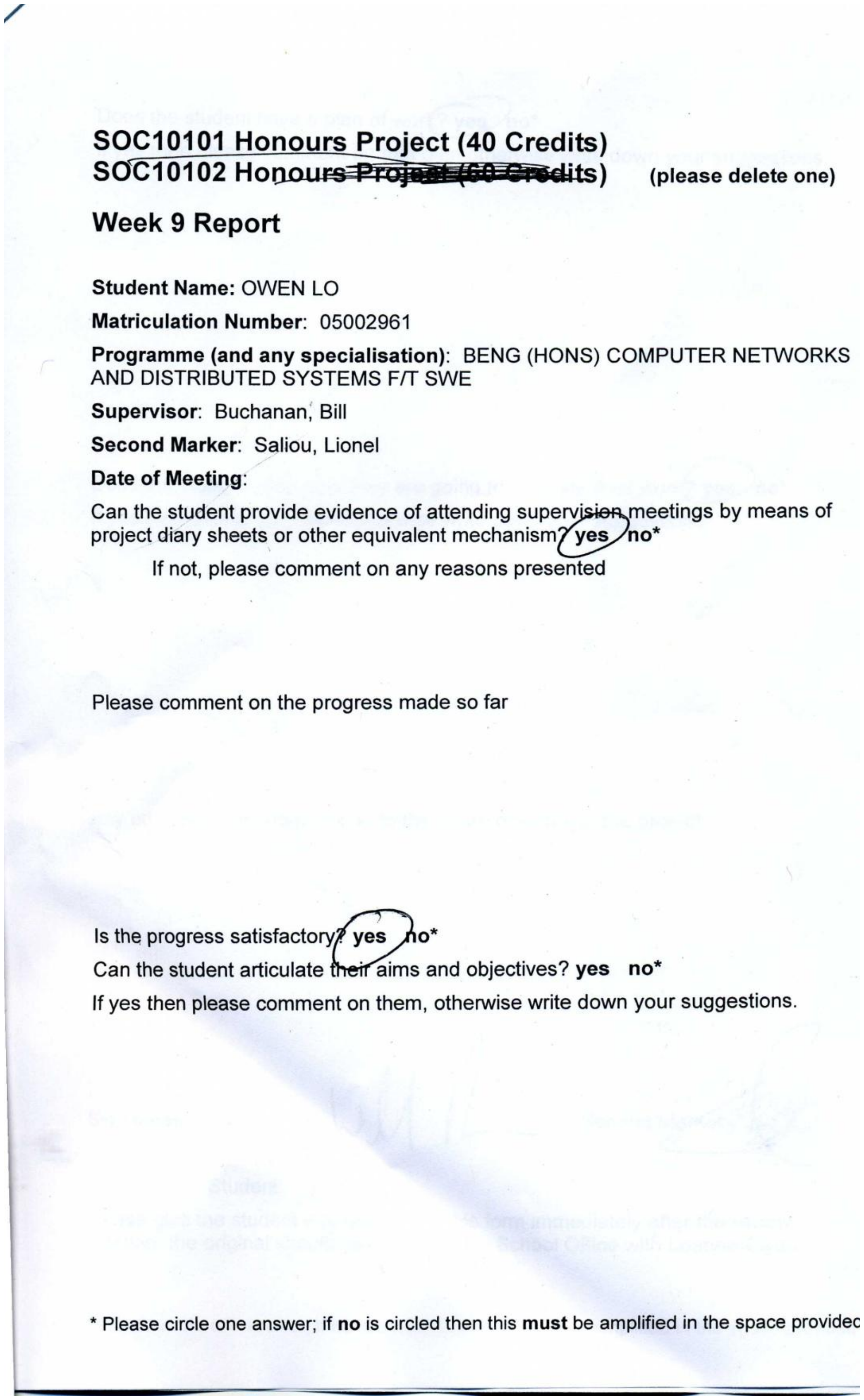
The main key challenge will be in deciding what the exact nature of the tool to be implemented for testing the IDS will be.

Another challenge will involve deciding what types of experiments will actually be carried out for this project.

Understanding and successfully implementing a method for performance metric generation will be another key challenge.

Lastly, acquiring in-depth knowledge in how IDS work will be another key challenge.

Appendix 2 Week 9 Meeting Report



SOC10101 Honours Project (40 Credits)
SOC10102 Honours ~~Project (60 Credits)~~ **(please delete one)**

Week 9 Report

Student Name: OWEN LO

Matriculation Number: 05002961

Programme (and any specialisation): BENG (HONS) COMPUTER NETWORKS AND DISTRIBUTED SYSTEMS F/T SWE

Supervisor: Buchanan, Bill

Second Marker: Saliou, Lionel

Date of Meeting:

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? [✓ **yes**] **no***

If not, please comment on any reasons presented

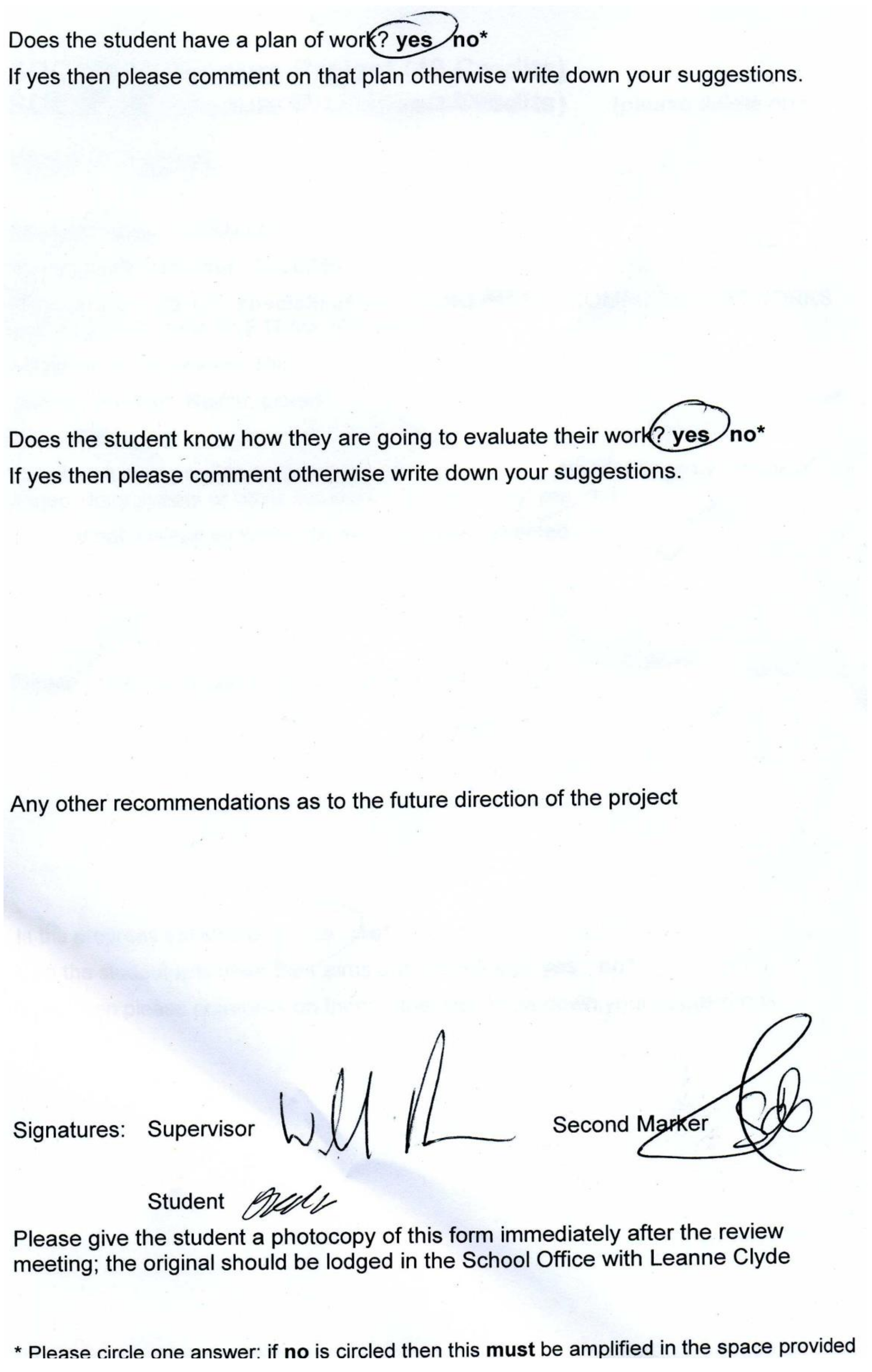
Please comment on the progress made so far

Is the progress satisfactory? [✓ **yes**] **no***

Can the student articulate their aims and objectives? **yes** **no***

If yes then please comment on them, otherwise write down your suggestions.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided



Does the student have a plan of work? [✓ yes] no*

If yes then please comment on that plan otherwise write down your suggestions.

Does the student know how they are going to evaluate their work? [✓ yes] no*

If yes then please comment otherwise write down your suggestions.

Any other recommendations as to the future direction of the project

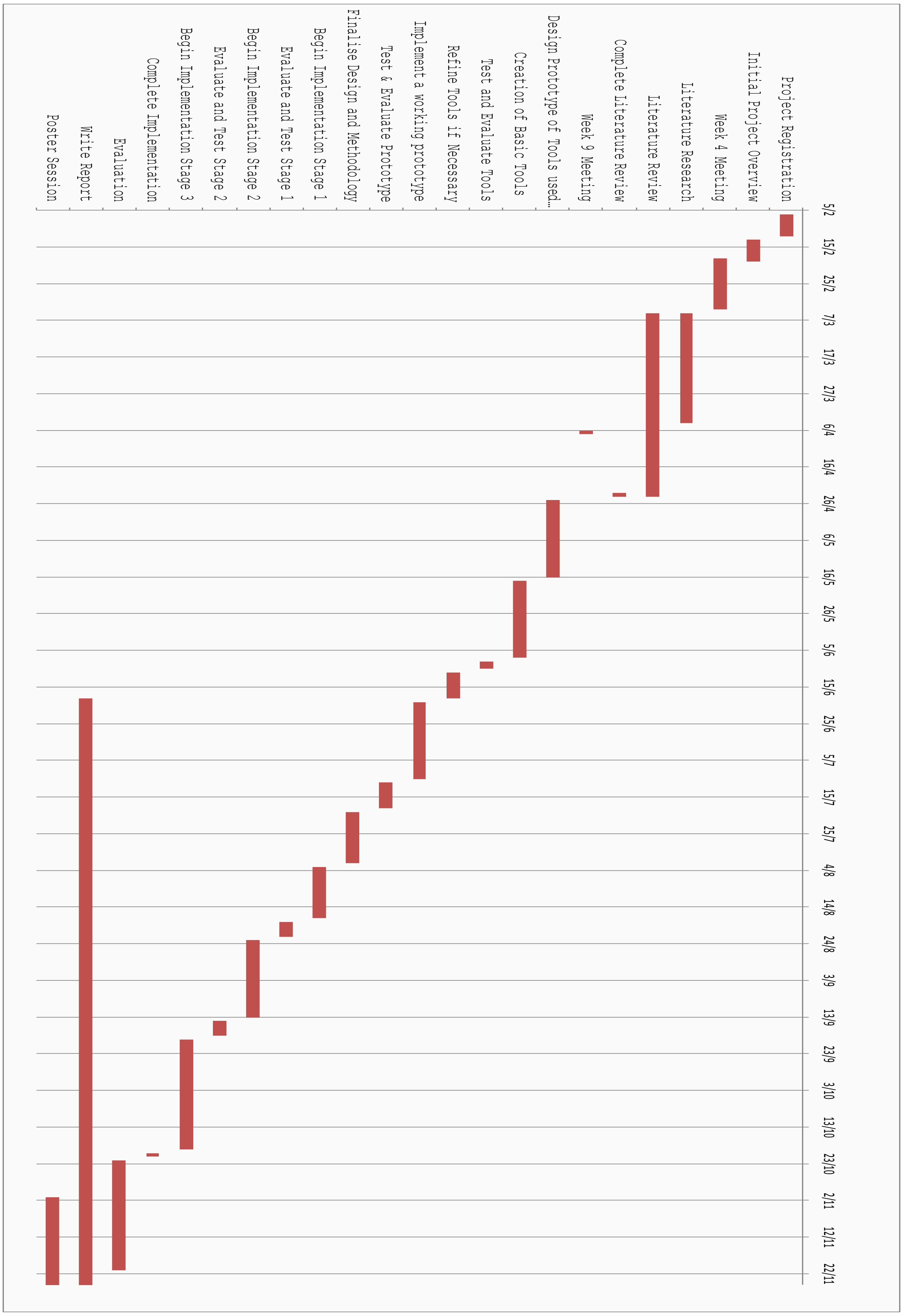
Signatures: Supervisor [illegible] Second Marker [illegible]

Student [illegible]

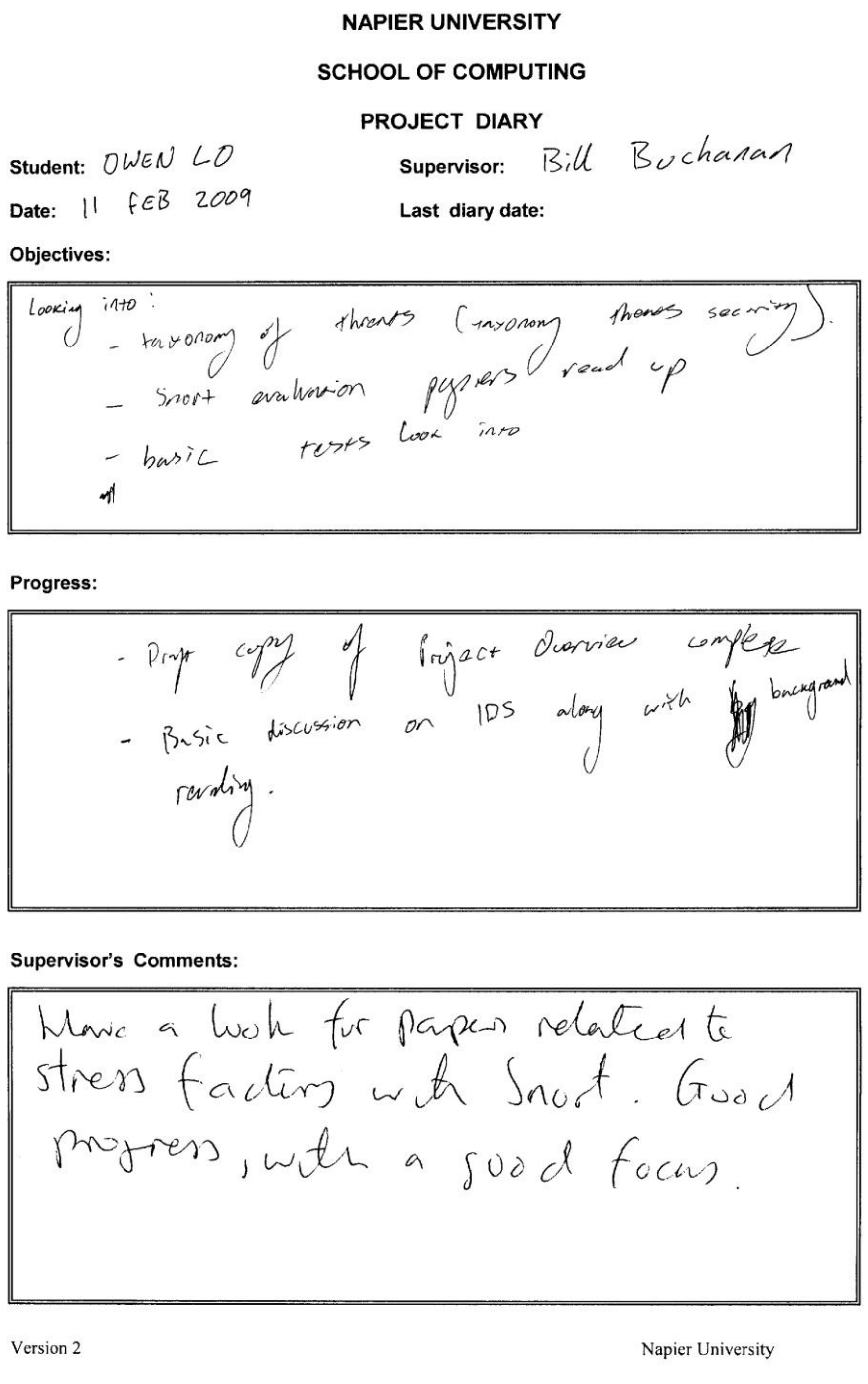
Please give the student a photocopy of this form immediately after the review meeting; the original should be lodged in the School Office with Leanne Clyde

* Please circle one answer: if **no** is circled then this **must** be amplified in the space provided

Appendix 3 Time Plan



Appendix 4 Diary Sheets



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student:	OWEN LO	**Supervisor:**	Bill Buchanan
Date:	11 FEB 2009	**Last diary date:**	

Objectives:

Looking into:
- taxonomy of threats (taxonomy threats security).
- Snort evaluation papers read up
- basic tests look into

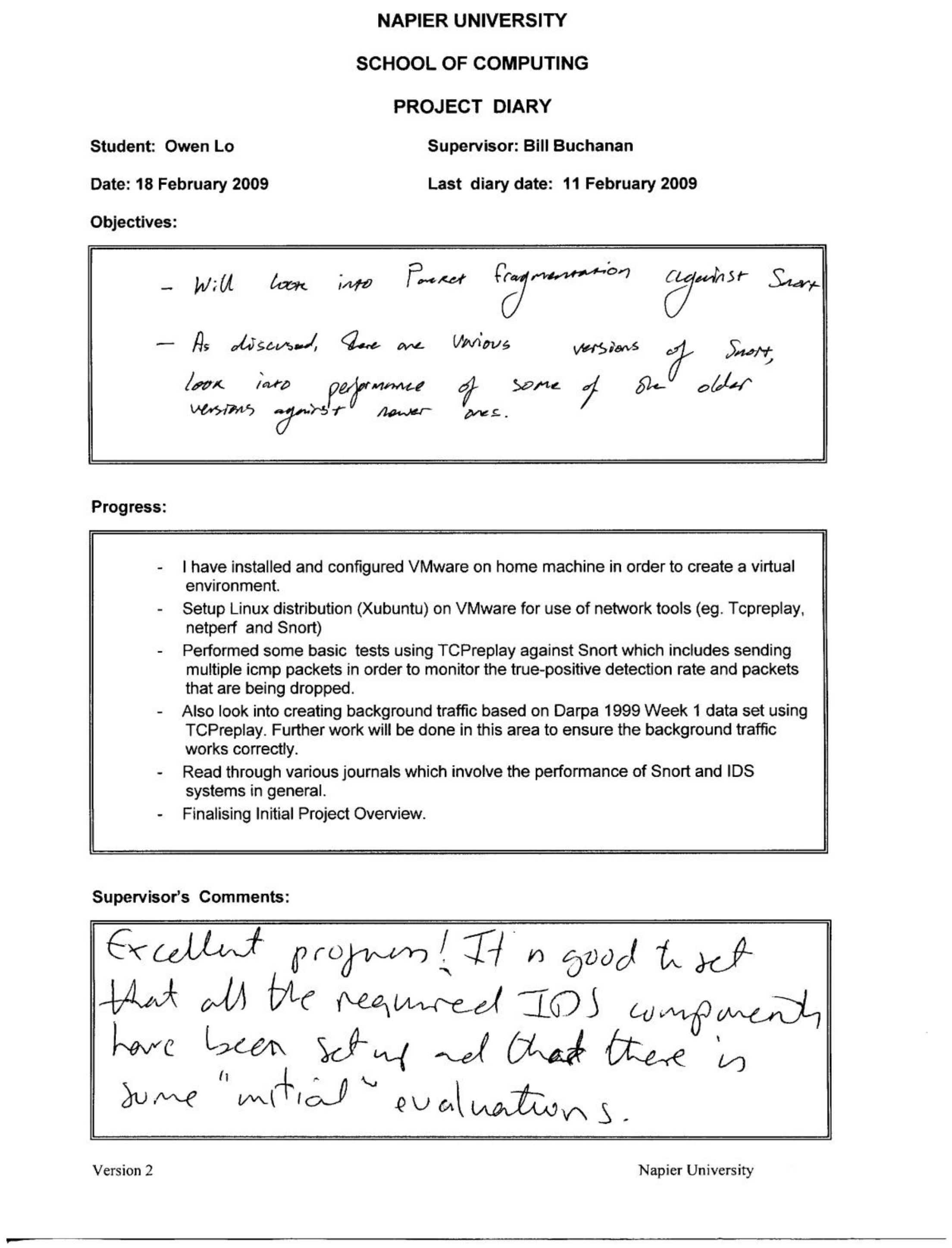
Progress:

- Print copy of Project Overview complete
- Basic discussion on IDS along with ~~[illegible]~~ background reading.

Supervisor's Comments:

Have a look for papers related to stress testing with Snort. Good progress, with a good focus.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 18 February 2009	**Last diary date: 11 February 2009**

Objectives:

- Will look into Packet fragmentation against Snort
- As discussed, there are various versions of Snort, look into performance of some of the older versions against newer ones.

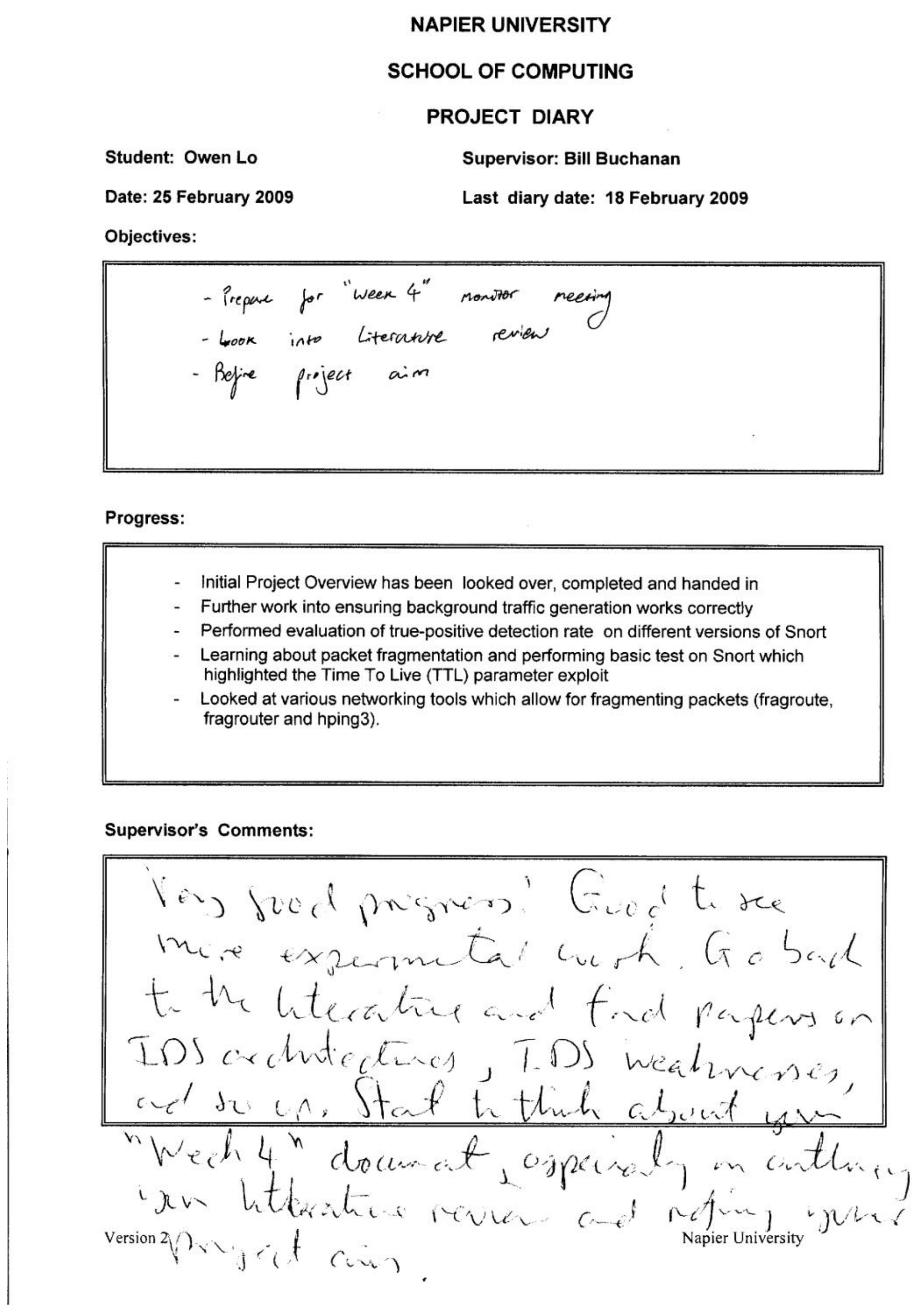
Progress:

- I have installed and configured VMware on home machine in order to create a virtual environment.
- Setup Linux distribution (Xubuntu) on VMware for use of network tools (eg. Tcpreplay, netperf and Snort)
- Performed some basic tests using TCPreplay against Snort which includes sending multiple icmp packets in order to monitor the true-positive detection rate and packets that are being dropped.
- Also look into creating background traffic based on Darpa 1999 Week 1 data set using TCPreplay. Further work will be done in this area to ensure the background traffic works correctly.
- Read through various journals which involve the performance of Snort and IDS systems in general.
- Finalising Initial Project Overview.

Supervisor's Comments:

Excellent progress! It is good to see that all the required IDS components have been set up and that there is some "initial" evaluations.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 25 February 2009	**Last diary date: 18 February 2009**

Objectives:

- Prepare for "Week 4" monitor meeting
- Look into Literature review
- Define project aim

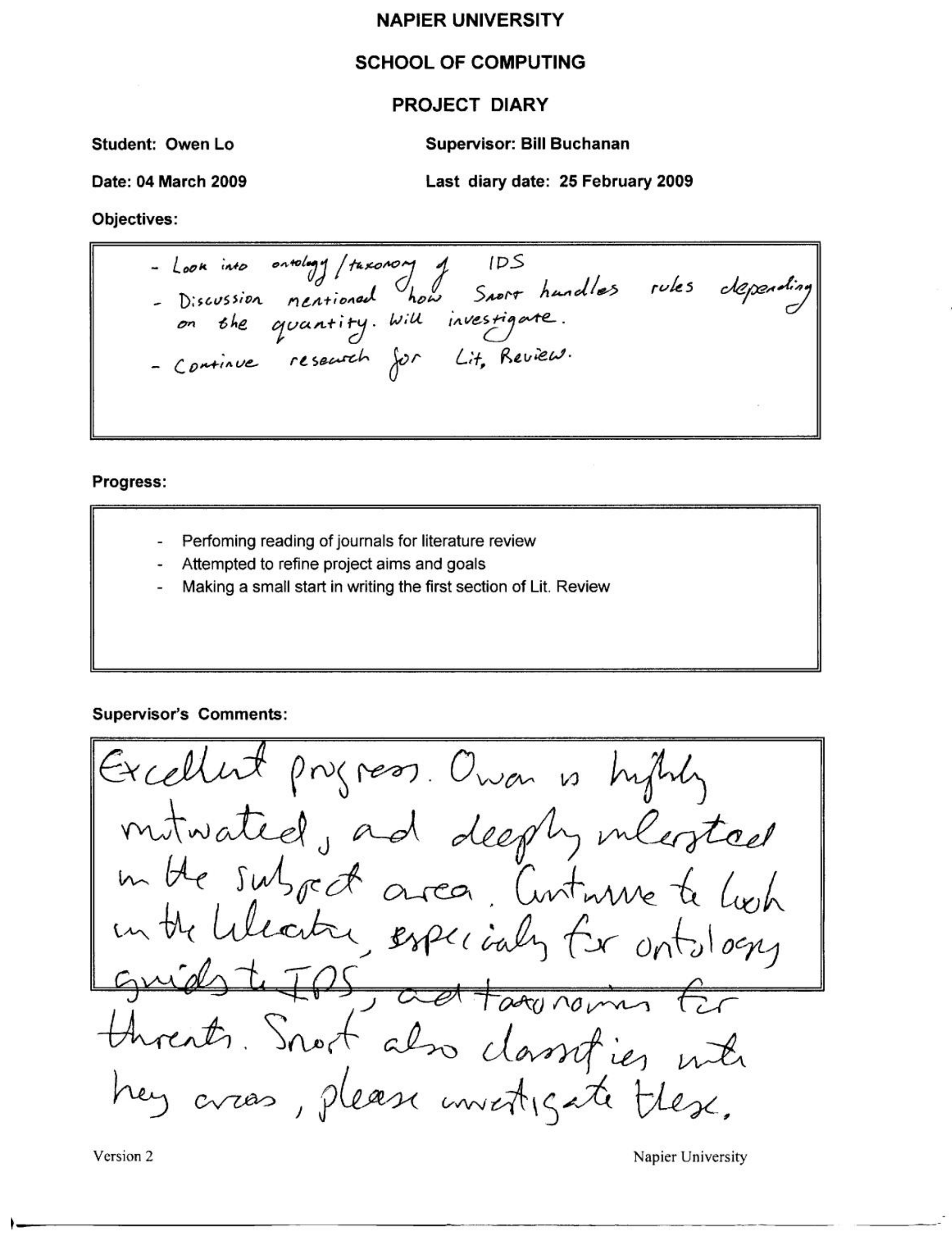
Progress:

- Initial Project Overview has been looked over, completed and handed in
- Further work into ensuring background traffic generation works correctly
- Performed evaluation of true-positive detection rate on different versions of Snort
- Learning about packet fragmentation and performing basic test on Snort which highlighted the Time To Live (TTL) parameter exploit
- Looked at various networking tools which allow for fragmenting packets (fragroute, fragrouter and hping3).

Supervisor's Comments:

Very good progress! Good to see more experimental work. Go back to the literature and find papers on IDS architectures, IDS weaknesses, and so on. Start to think about your "Week 4" document, especially on outlining your literature review and refining your project aim.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo **Supervisor: Bill Buchanan**

Date: 04 March 2009 **Last diary date: 25 February 2009**

Objectives:

- Look into ontology / taxonomy of IDS
- Discussion mentioned how Snort handles rules depending on the quantity. Will investigate.
- Continue research for Lit. Review.

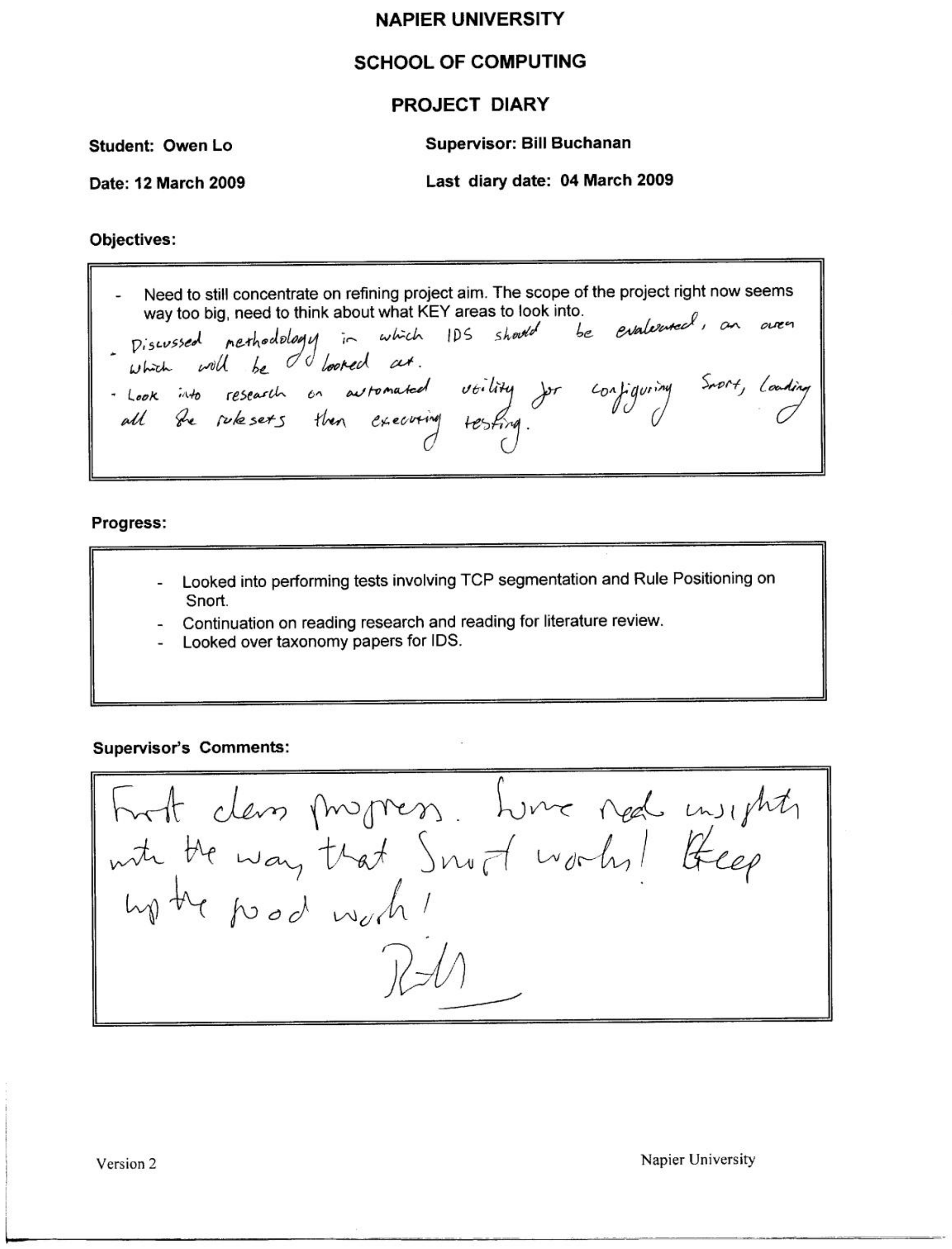
Progress:

- Perfoming reading of journals for literature review
- Attempted to refine project aims and goals
- Making a small start in writing the first section of Lit. Review

Supervisor's Comments:

Excellent progress. Owen is highly motivated, and deeply understood in the subject area. Continue to look in the literature, especially for ontology guides to IDS, and taxonomies for threats. Snort also classifies into key areas, please investigate these.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 12 March 2009	**Last diary date: 04 March 2009**

Objectives:

- Need to still concentrate on refining project aim. The scope of the project right now seems way too big, need to think about what KEY areas to look into.
- Discussed methodology in which IDS should be evaluated, an area which will be looked at.
- Look into research on automated utility for configuring Snort, loading all the rulesets then executing testing.

Progress:

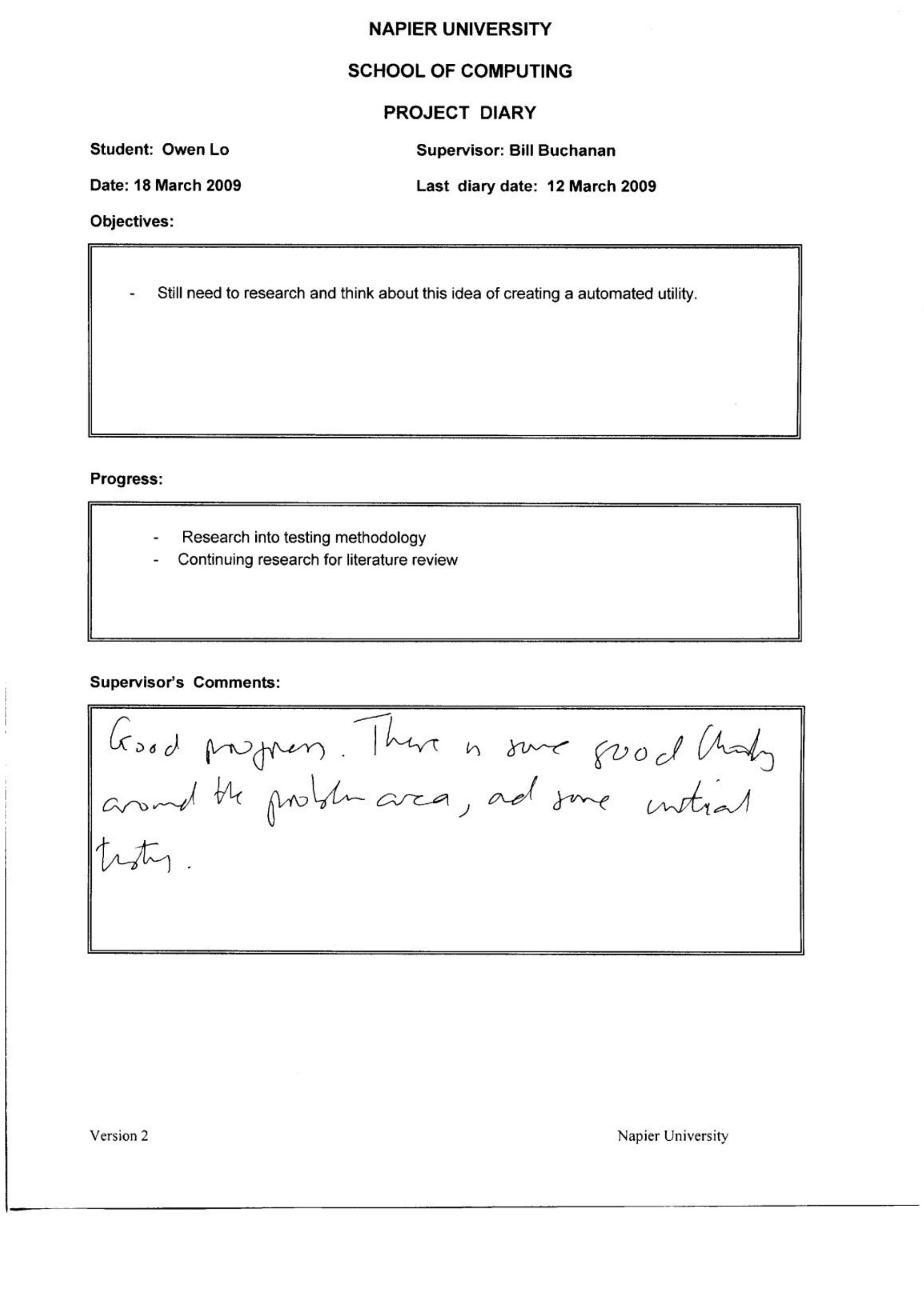
- Looked into performing tests involving TCP segmentation and Rule Positioning on Snort.
- Continuation on reading research and reading for literature review.
- Looked over taxonomy papers for IDS.

Supervisor's Comments:

First class progress. Some real insights into the way that Snort works! Keep up the good work!

[illegible]

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 18 March 2009	**Last diary date: 12 March 2009**

Objectives:

- Still need to research and think about this idea of creating a automated utility.

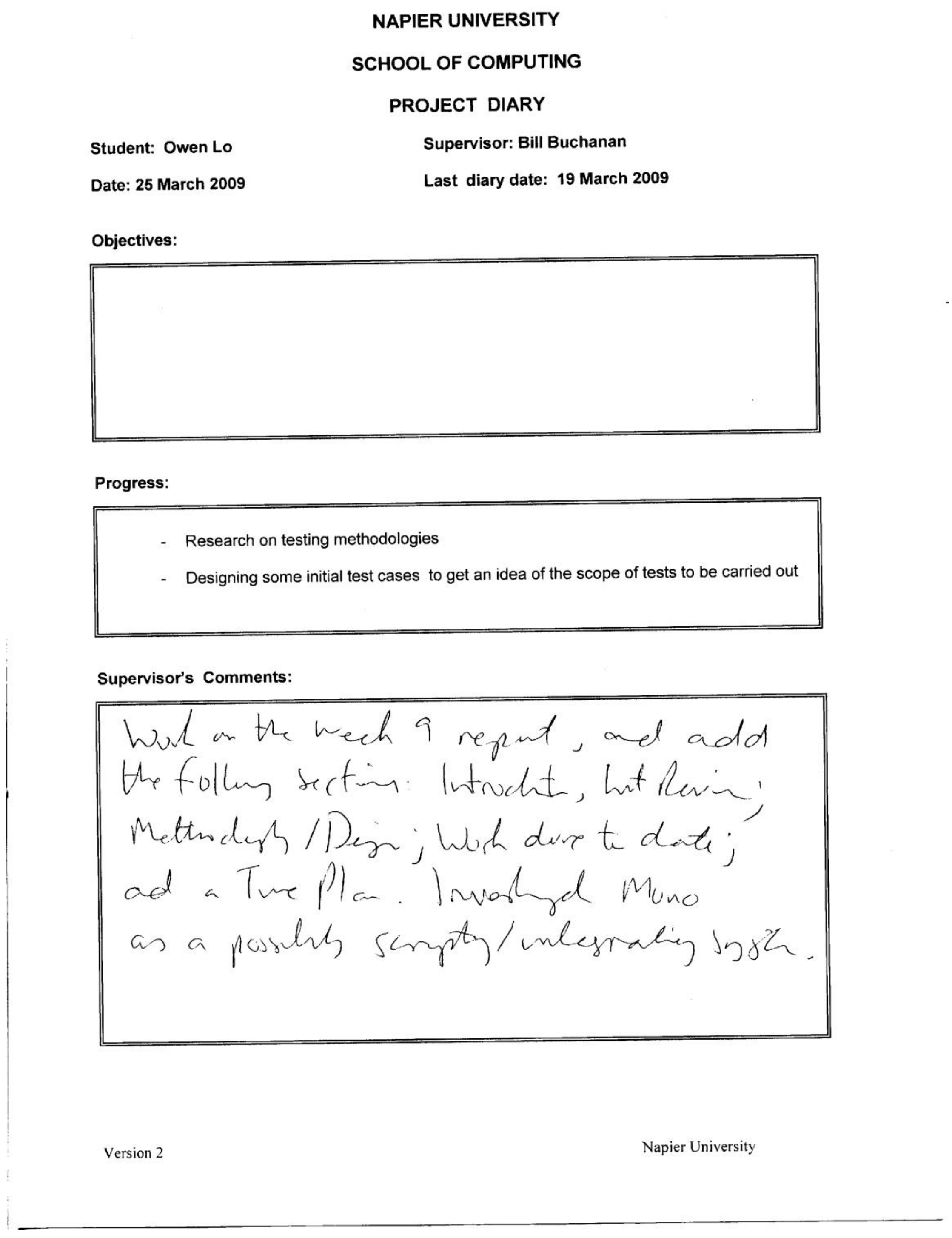
Progress:

- Research into testing methodology
- Continuing research for literature review

Supervisor's Comments:

Good progress. There is some good thinking around the problem area, and some initial testing.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	Supervisor: Bill Buchanan
Date: 25 March 2009	Last diary date: 19 March 2009

Objectives:

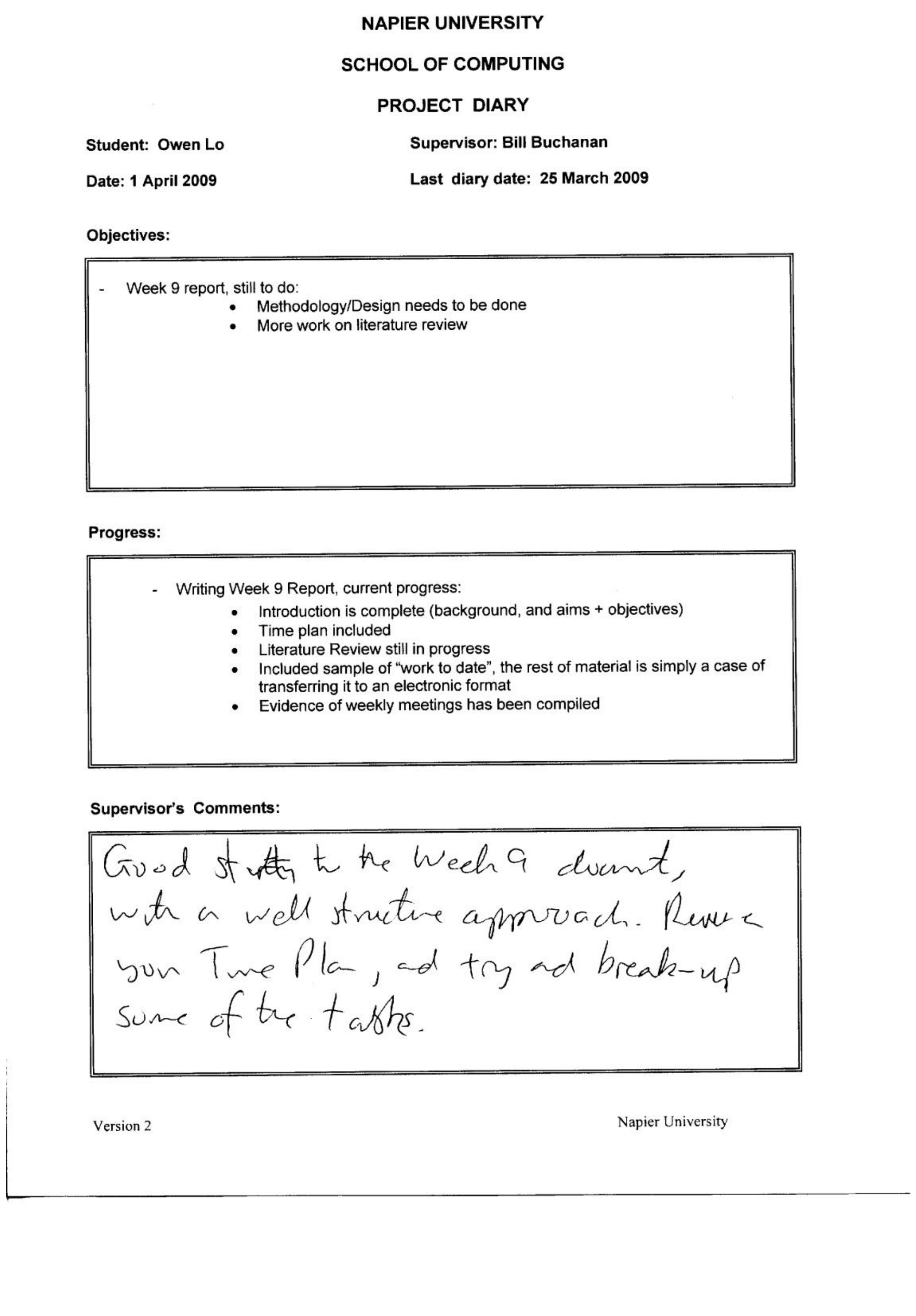
Progress:

- Research on testing methodologies
- Designing some initial test cases to get an idea of the scope of tests to be carried out

Supervisor's Comments:

Work on the week 9 report, and add the following sections: Introduction, Lit Review; Methodology / Design; Work done to date; and a Time Plan. Investigated Mono as a possibility scripting / integrating system.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 1 April 2009	**Last diary date: 25 March 2009**

Objectives:

- Week 9 report, still to do:
    - Methodology/Design needs to be done
    - More work on literature review

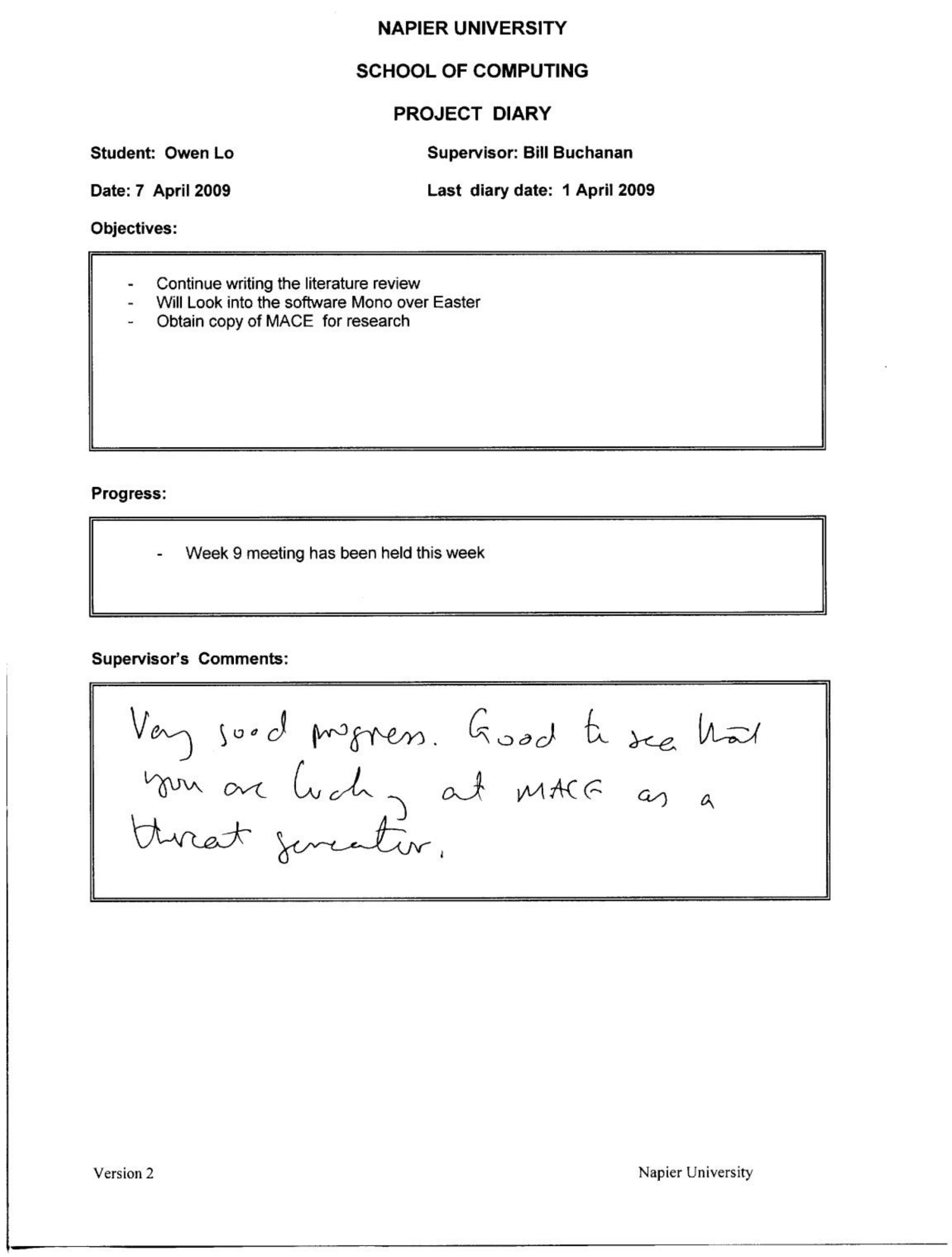
Progress:

- Writing Week 9 Report, current progress:
    - Introduction is complete (background, and aims + objectives)
    - Time plan included
    - Literature Review still in progress
    - Included sample of "work to date", the rest of material is simply a case of transferring it to an electronic format
    - Evidence of weekly meetings has been compiled

Supervisor's Comments:

Good st[illegible] to the Week 9 document, with a well structure approach. Review your Time Plan, and try and break-up some of the tasks.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo **Supervisor: Bill Buchanan**

Date: 7 April 2009 **Last diary date: 1 April 2009**

Objectives:

- Continue writing the literature review
- Will Look into the software Mono over Easter
- Obtain copy of MACE for research

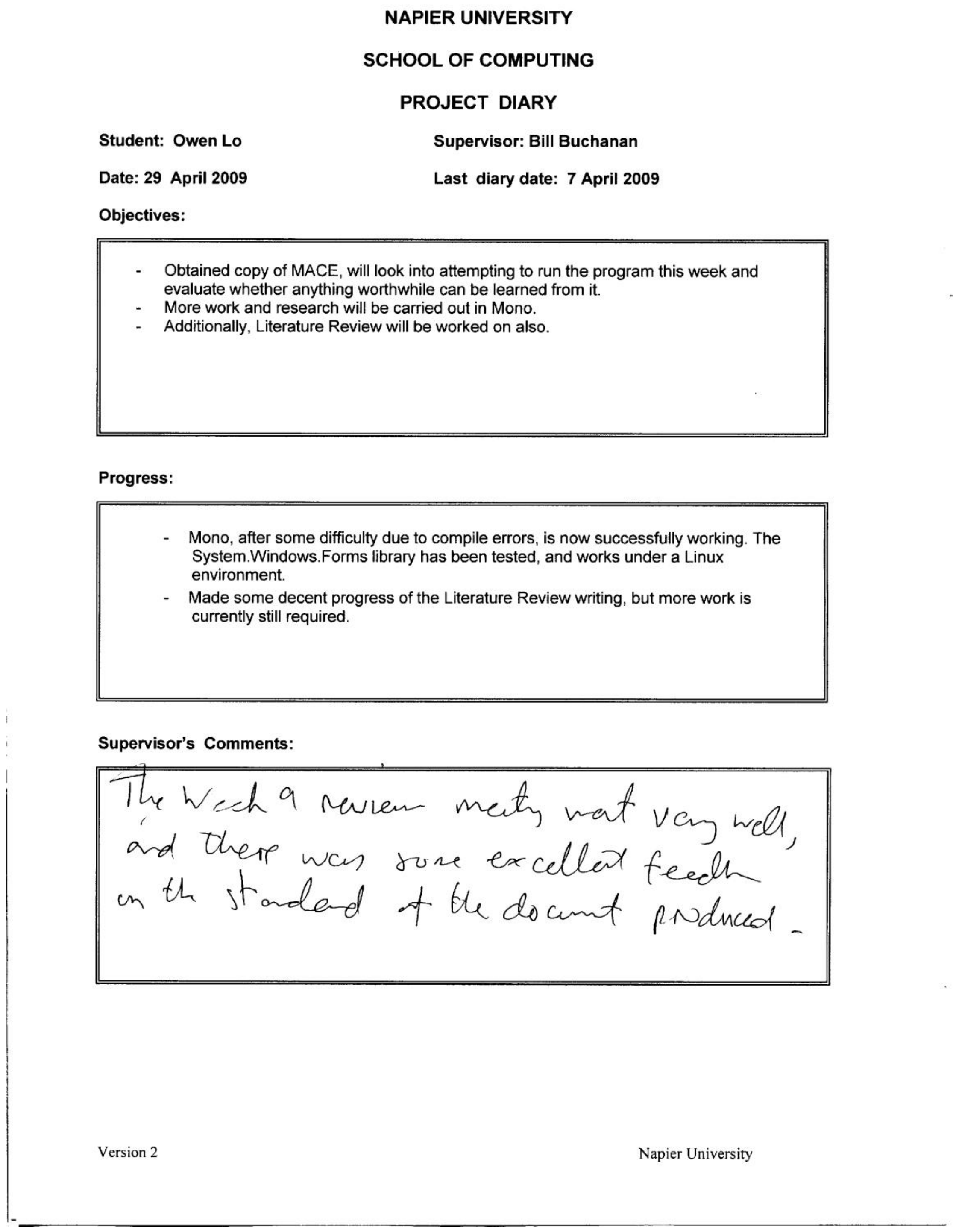
Progress:

- Week 9 meeting has been held this week

Supervisor's Comments:

Very good progress. Good to see that you are looking at MACE as a threat generator.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 29 April 2009	**Last diary date: 7 April 2009**

Objectives:

- Obtained copy of MACE, will look into attempting to run the program this week and evaluate whether anything worthwhile can be learned from it.
- More work and research will be carried out in Mono.
- Additionally, Literature Review will be worked on also.

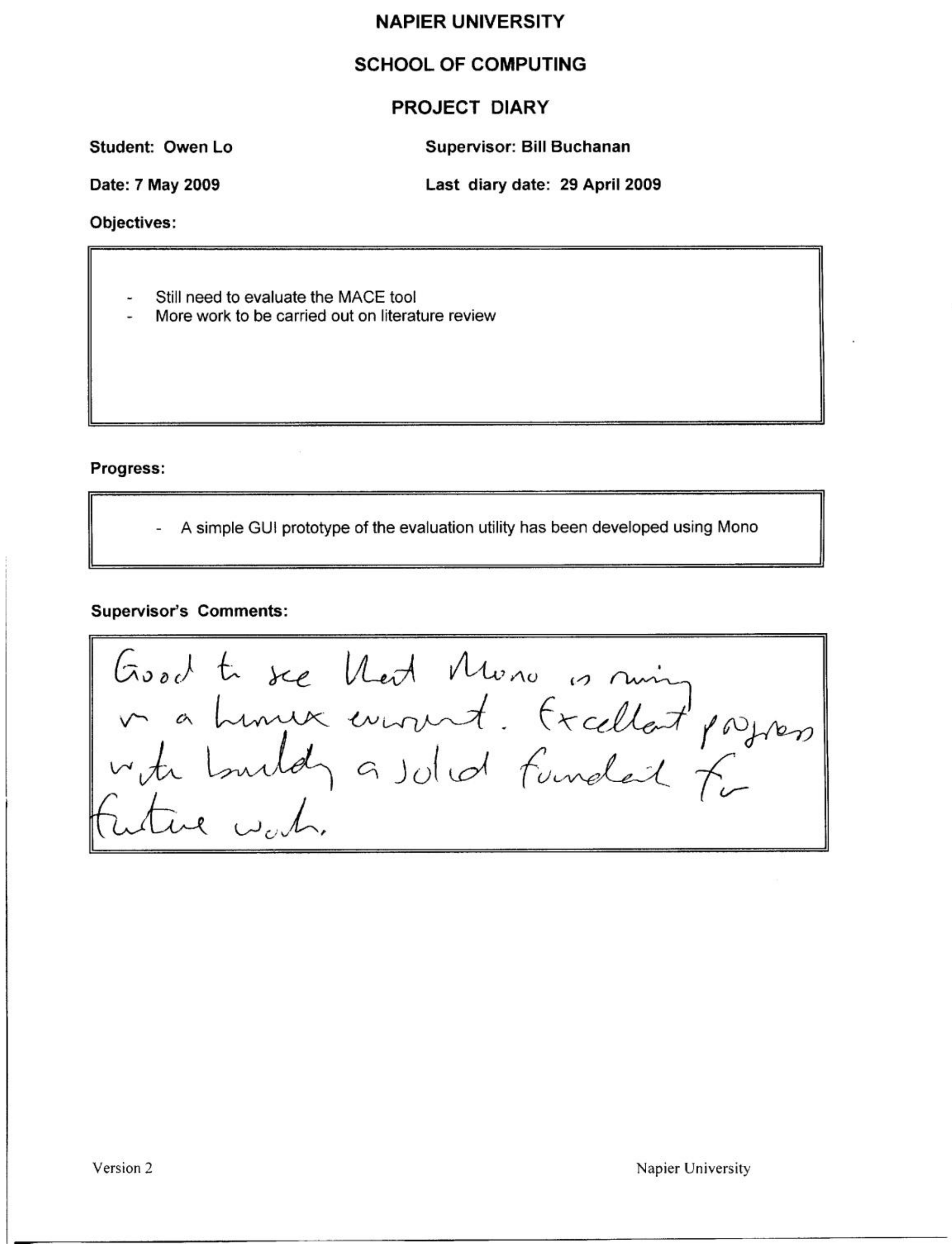
Progress:

- Mono, after some difficulty due to compile errors, is now successfully working. The System.Windows.Forms library has been tested, and works under a Linux environment.
- Made some decent progress of the Literature Review writing, but more work is currently still required.

Supervisor's Comments:

The Week 9 review meeting went very well, and there was some excellent feedb on the standard of the document produced.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 7 May 2009	**Last diary date: 29 April 2009**

Objectives:

- Still need to evaluate the MACE tool
- More work to be carried out on literature review

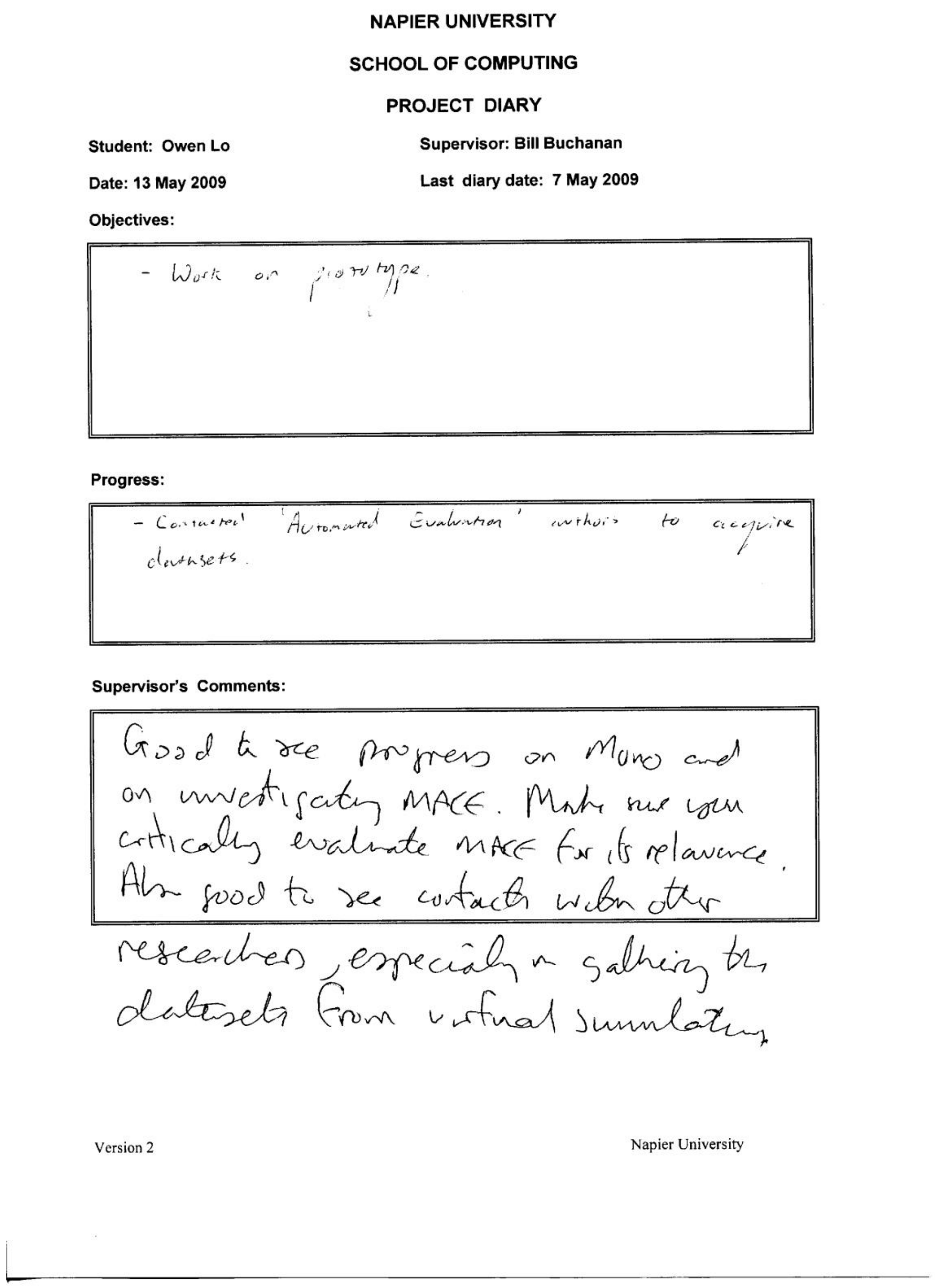
Progress:

- A simple GUI prototype of the evaluation utility has been developed using Mono

Supervisor's Comments:

Good to see that Mono is running on a Linux environment. Excellent progress with building a solid foundation for future work.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 13 May 2009	**Last diary date: 7 May 2009**

Objectives:

- Work on prototype.

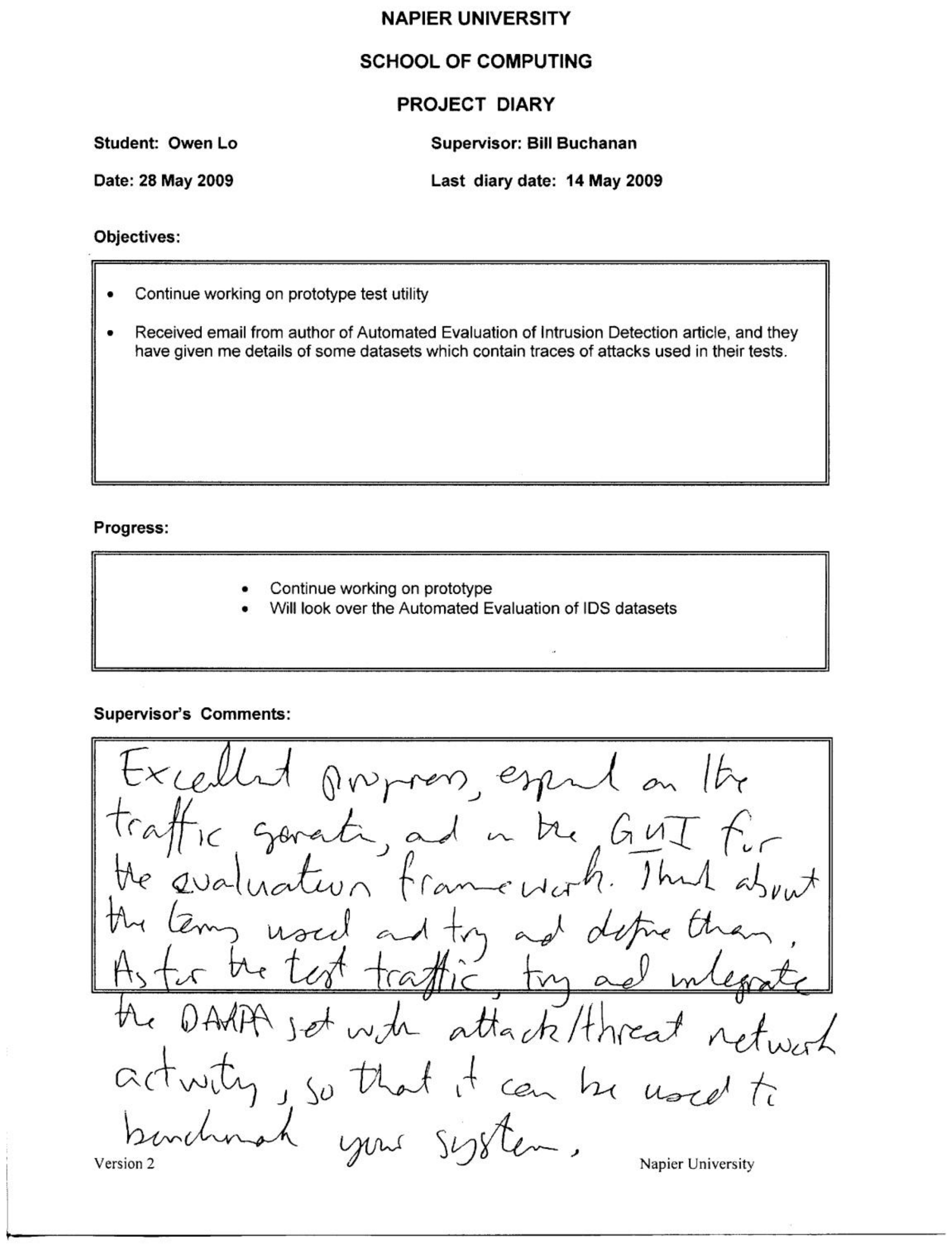
Progress:

- Contacted 'Automated Evaluation' authors to acquire datasets.

Supervisor's Comments:

Good to see progress on Mono and on investigating MACE. Make sure you critically evaluate MACE for its relavence. Also good to see contacts with other researchers, especially in gathering the datasets from virtual simulating.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 28 May 2009	**Last diary date: 14 May 2009**

Objectives:

- Continue working on prototype test utility
- Received email from author of Automated Evaluation of Intrusion Detection article, and they have given me details of some datasets which contain traces of attacks used in their tests.

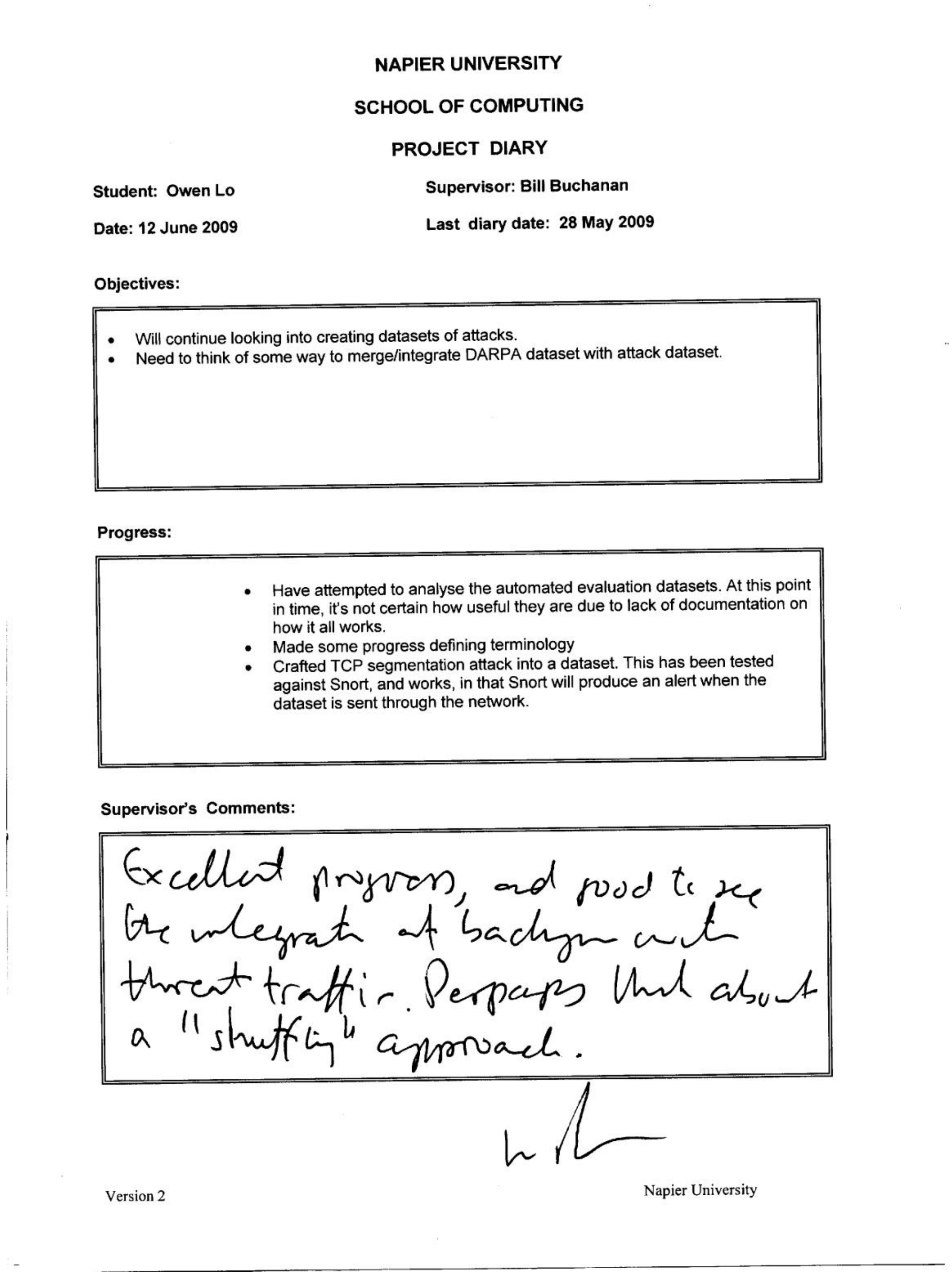
Progress:

- Continue working on prototype
- Will look over the Automated Evaluation of IDS datasets

Supervisor's Comments:

Excellent progress, espcl on the traffic generator, and on the GUI for the evaluation framework. Think about the terms used and try and define them. As for the test traffic, try and integrate the DARPA set with attack/threat network activity, so that it can be used to benchmark your system.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo **Supervisor:** Bill Buchanan

Date: 12 June 2009 **Last diary date:** 28 May 2009

Objectives:

- Will continue looking into creating datasets of attacks.
- Need to think of some way to merge/integrate DARPA dataset with attack dataset.

Progress:

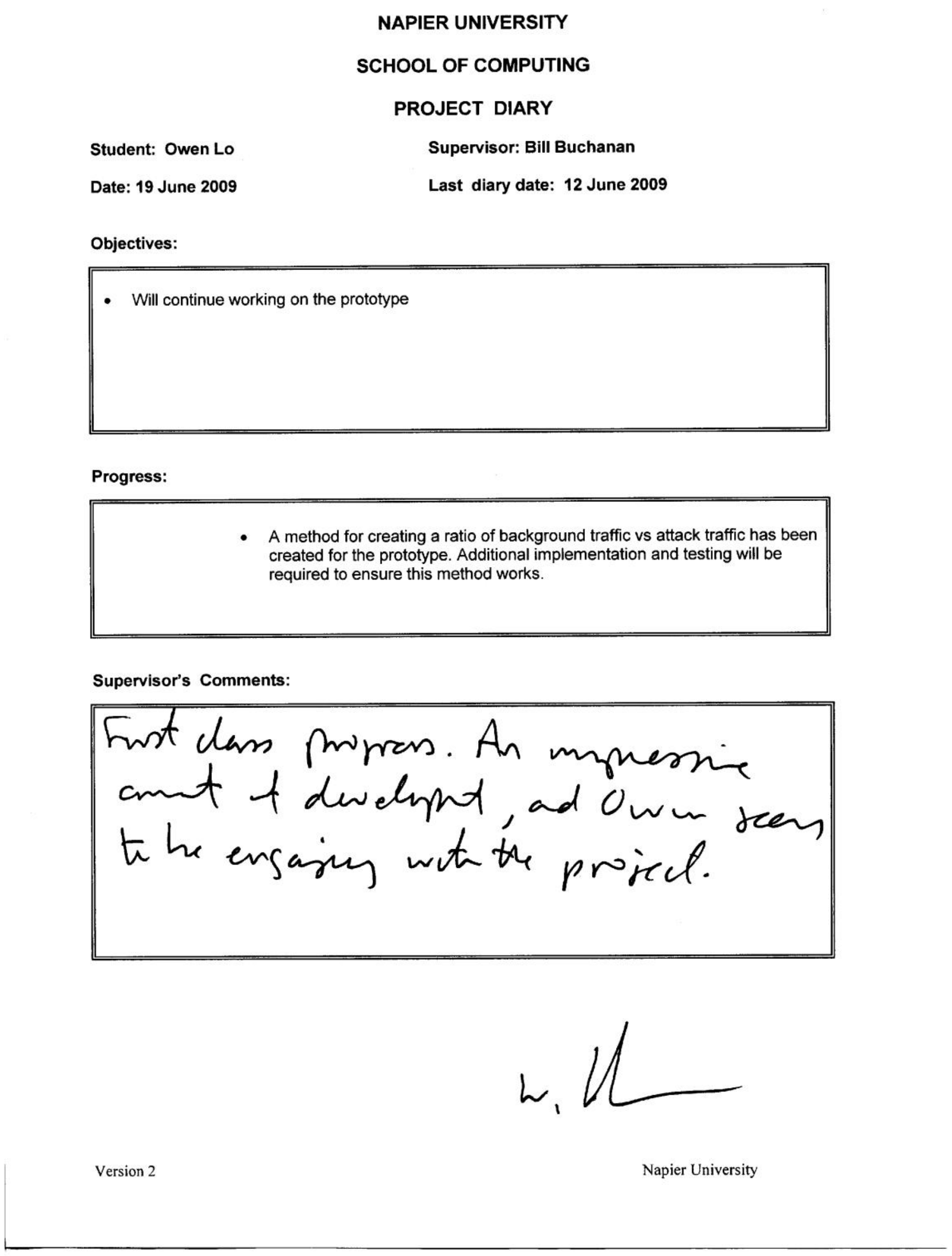
- Have attempted to analyse the automated evaluation datasets. At this point in time, it's not certain how useful they are due to lack of documentation on how it all works.
- Made some progress defining terminology
- Crafted TCP segmentation attack into a dataset. This has been tested against Snort, and works, in that Snort will produce an alert when the dataset is sent through the network.

Supervisor's Comments:

Excellent progress, and good to see the integration of background with threat traffic. Perhaps think about a "shuffling" approach.

[illegible]

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 19 June 2009	**Last diary date: 12 June 2009**

Objectives:

- Will continue working on the prototype

Progress:

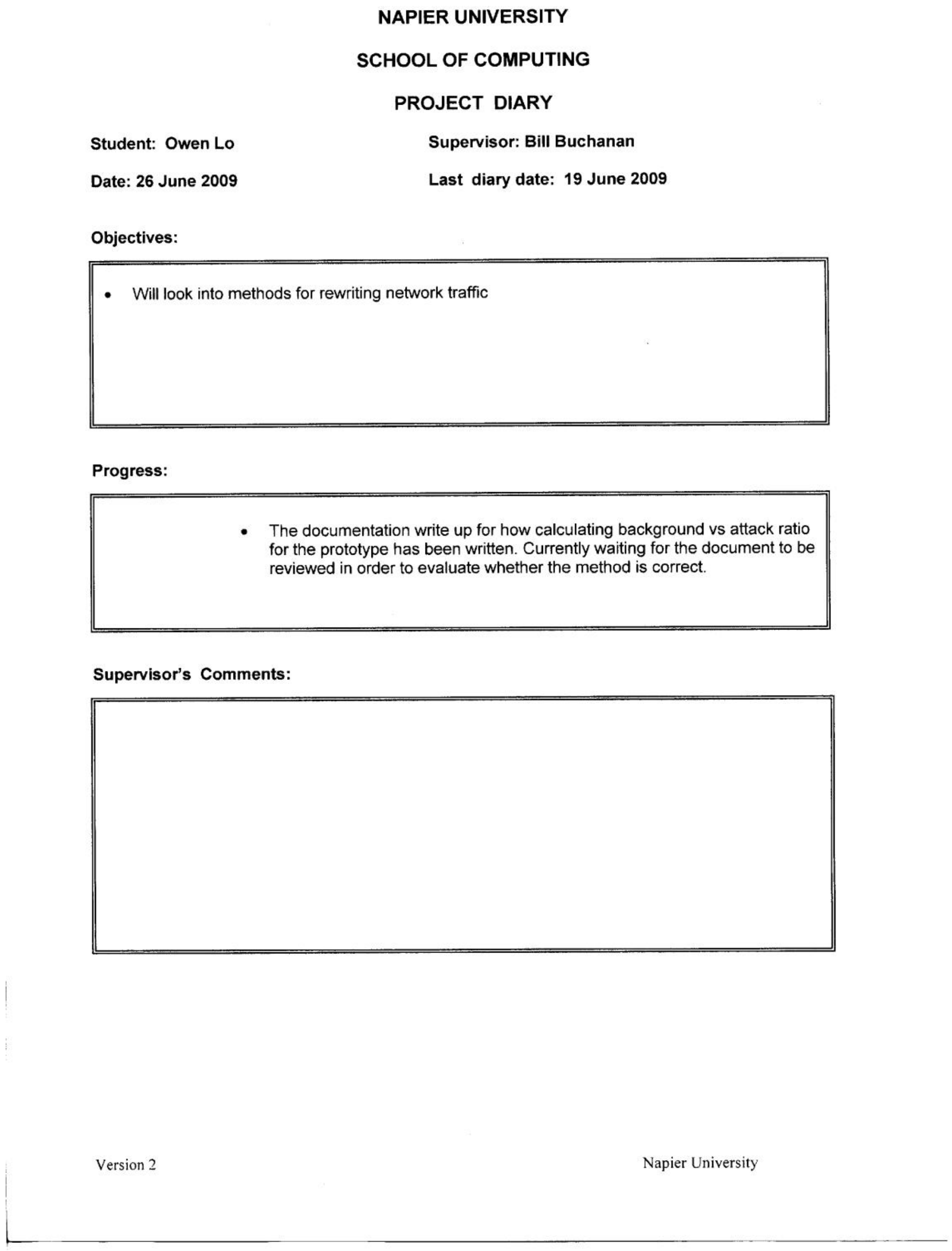
- A method for creating a ratio of background traffic vs attack traffic has been created for the prototype. Additional implementation and testing will be required to ensure this method works.

Supervisor's Comments:

First class progress. An impressive amount of development, and Owen seems to be engaging with the project.

[illegible]

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor:** Bill Buchanan
Date: 26 June 2009	**Last diary date:** 19 June 2009

Objectives:

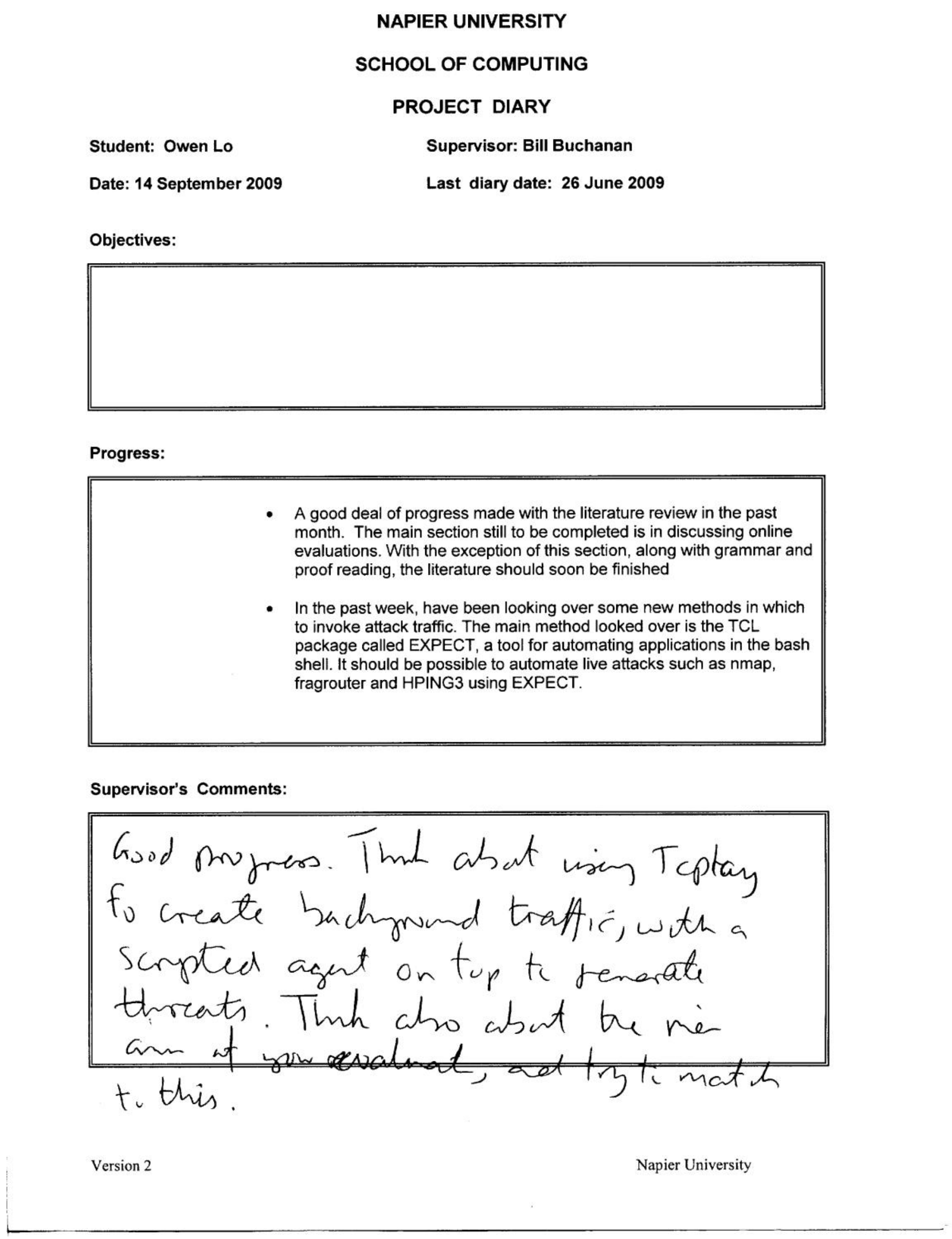
- Will look into methods for rewriting network traffic

Progress:

- The documentation write up for how calculating background vs attack ratio for the prototype has been written. Currently waiting for the document to be reviewed in order to evaluate whether the method is correct.

Supervisor's Comments:

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 14 September 2009	**Last diary date: 26 June 2009**

Objectives:

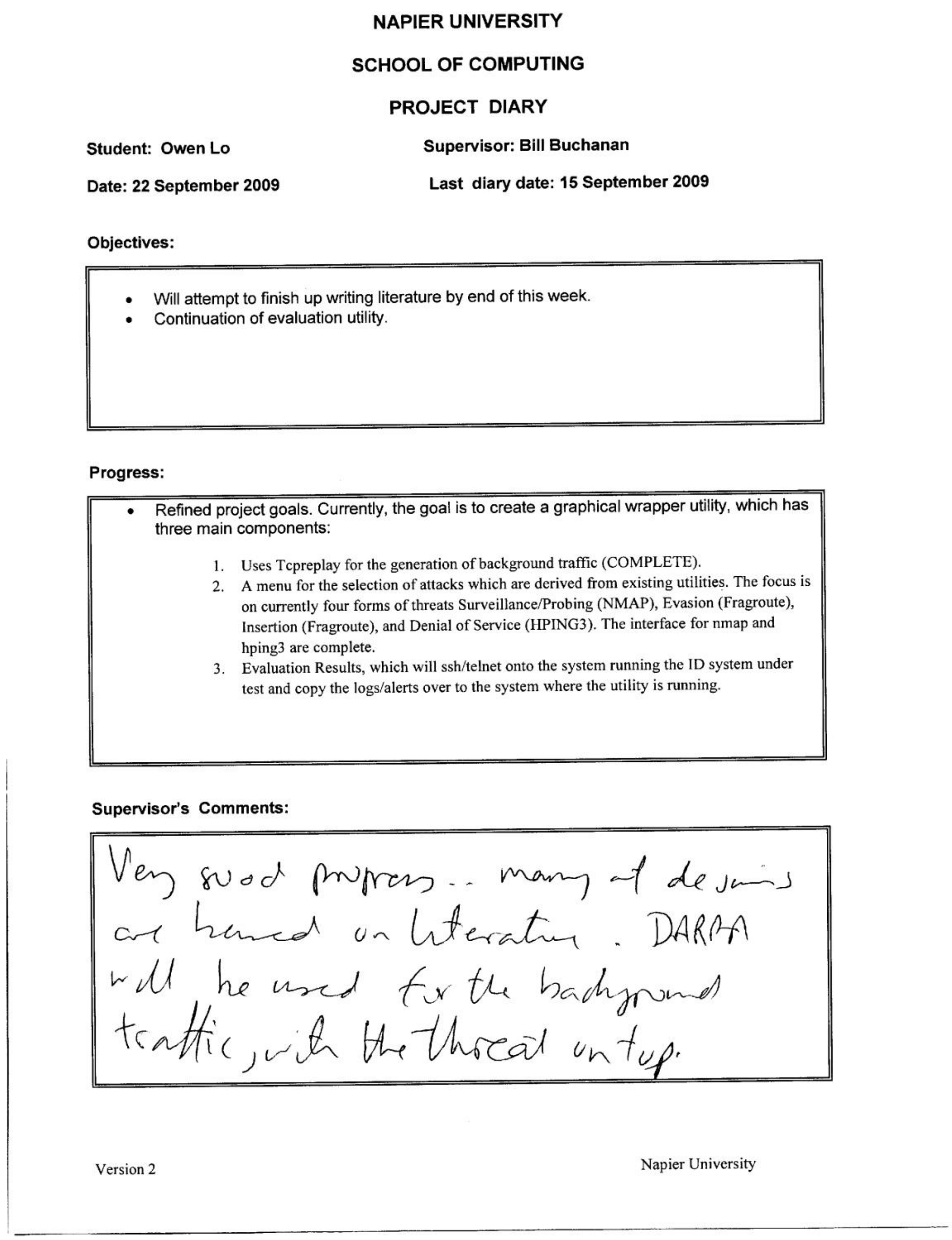
Progress:

- A good deal of progress made with the literature review in the past month. The main section still to be completed is in discussing online evaluations. With the exception of this section, along with grammar and proof reading, the literature should soon be finished
- In the past week, have been looking over some new methods in which to invoke attack traffic. The main method looked over is the TCL package called EXPECT, a tool for automating applications in the bash shell. It should be possible to automate live attacks such as nmap, fragrouter and HPING3 using EXPECT.

Supervisor's Comments:

Good progress. Think about using Tcplay to create background traffic, with a scripted agent on top to generate threats. Think also about the main aim of your evaluation, and try to match to this.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 22 September 2009	**Last diary date: 15 September 2009**

Objectives:

- Will attempt to finish up writing literature by end of this week.
- Continuation of evaluation utility.

Progress:

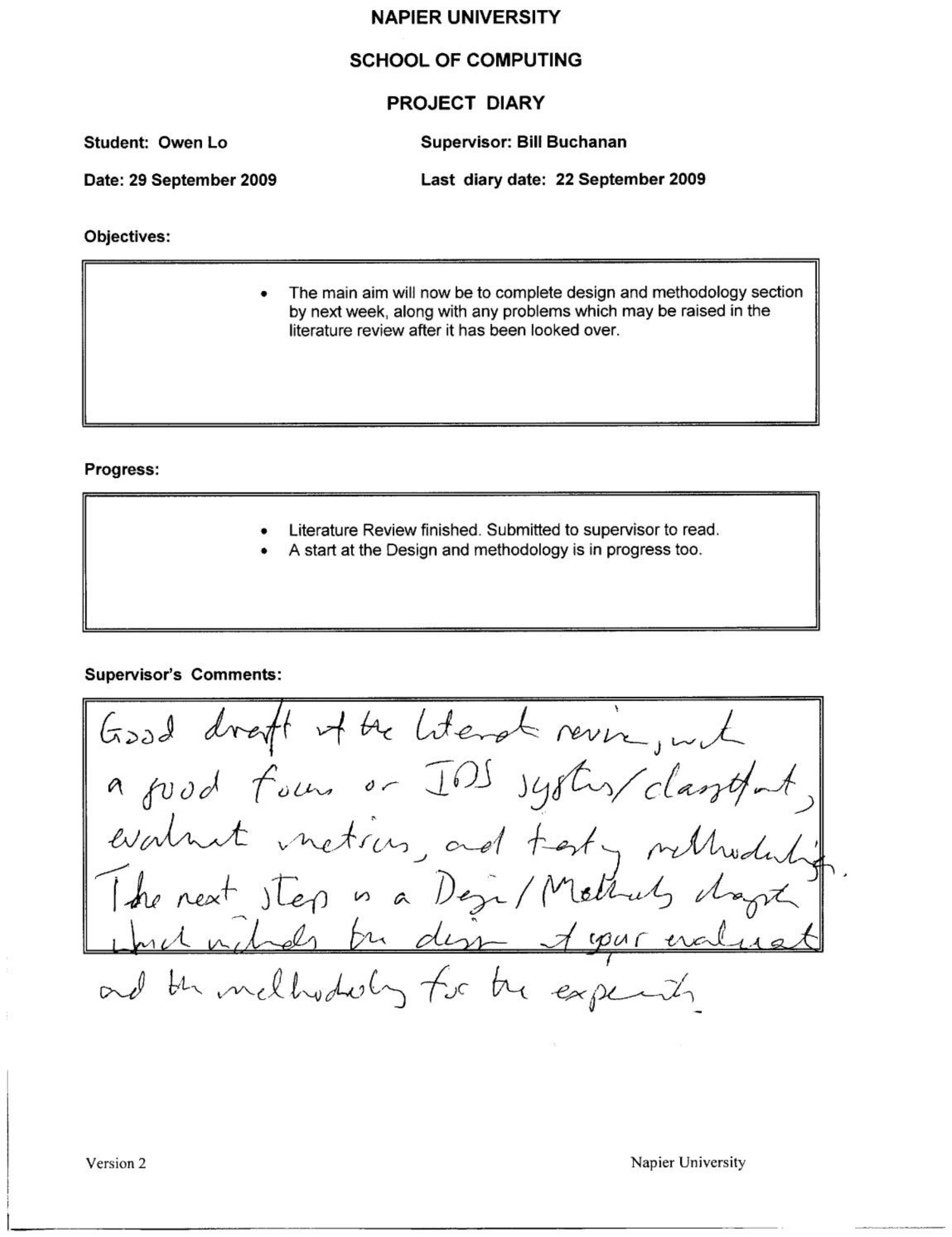
- Refined project goals. Currently, the goal is to create a graphical wrapper utility, which has three main components:

  1. Uses Tcpreplay for the generation of background traffic (COMPLETE).
  2. A menu for the selection of attacks which are derived from existing utilities. The focus is on currently four forms of threats Surveillance/Probing (NMAP), Evasion (Fragroute), Insertion (Fragroute), and Denial of Service (HPING3). The interface for nmap and hping3 are complete.
  3. Evaluation Results, which will ssh/telnet onto the system running the ID system under test and copy the logs/alerts over to the system where the utility is running.

Supervisor's Comments:

Very good progress.. many of designs are based on literature. DARPA will be used for the background traffic, with the threat on top.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo **Supervisor: Bill Buchanan**

Date: 29 September 2009 **Last diary date: 22 September 2009**

Objectives:

- The main aim will now be to complete design and methodology section by next week, along with any problems which may be raised in the literature review after it has been looked over.

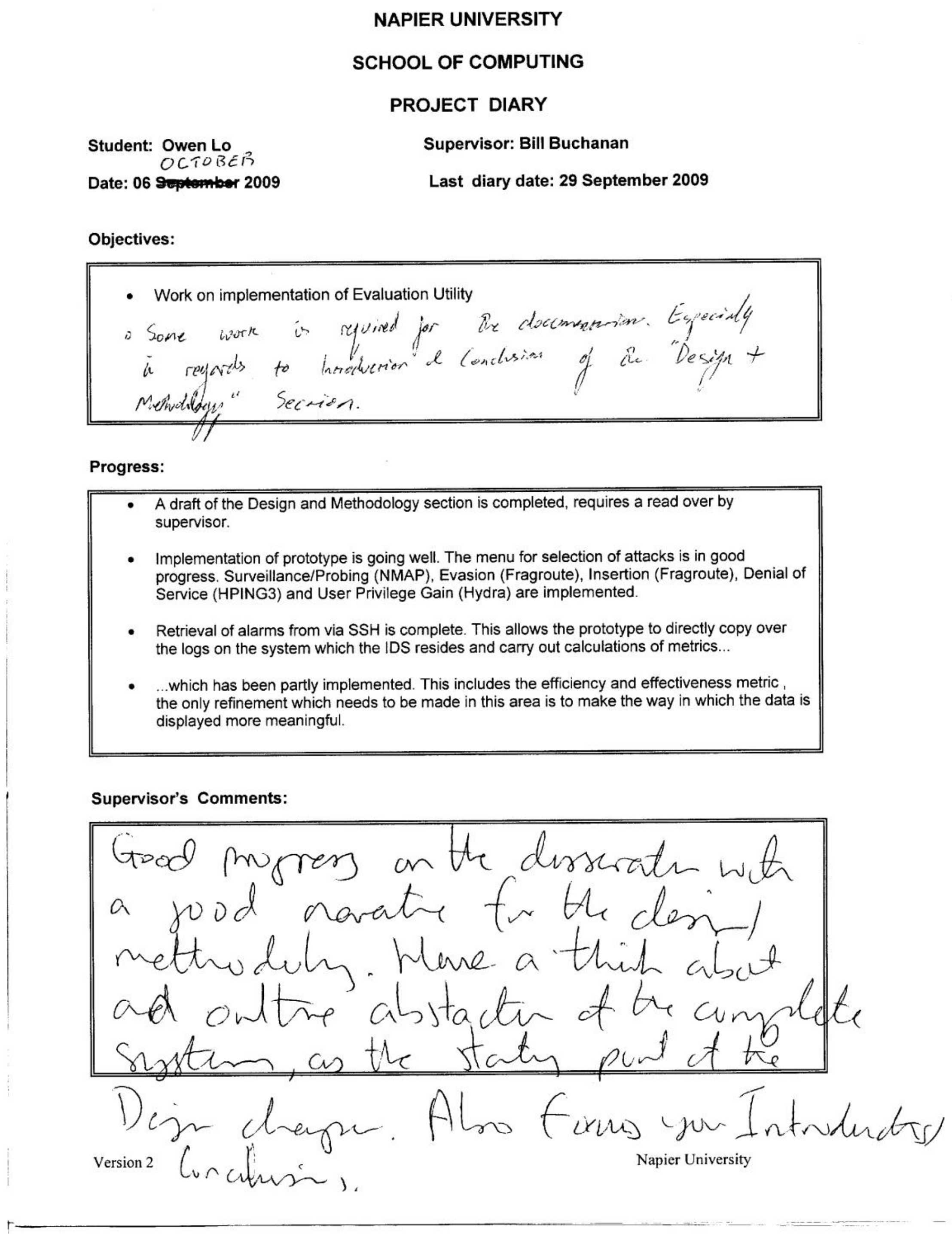
Progress:

- Literature Review finished. Submitted to supervisor to read.
- A start at the Design and methodology is in progress too.

Supervisor's Comments:

Good draft of the literature review, with a good focus on IDS systems/classification, evaluation metrics, and testing methodology. The next step is a Design/Methods chapter which includes the design of your evaluation and the methodology for the experiments.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo **Supervisor: Bill Buchanan**

Date: 06 ~~September~~ OCTOBER 2009 **Last diary date: 29 September 2009**

Objectives:

- Work on implementation of Evaluation Utility
- Some work is required for the documentation. Especially in regards to Introduction & Conclusion of the "Design + Methodology" Section.

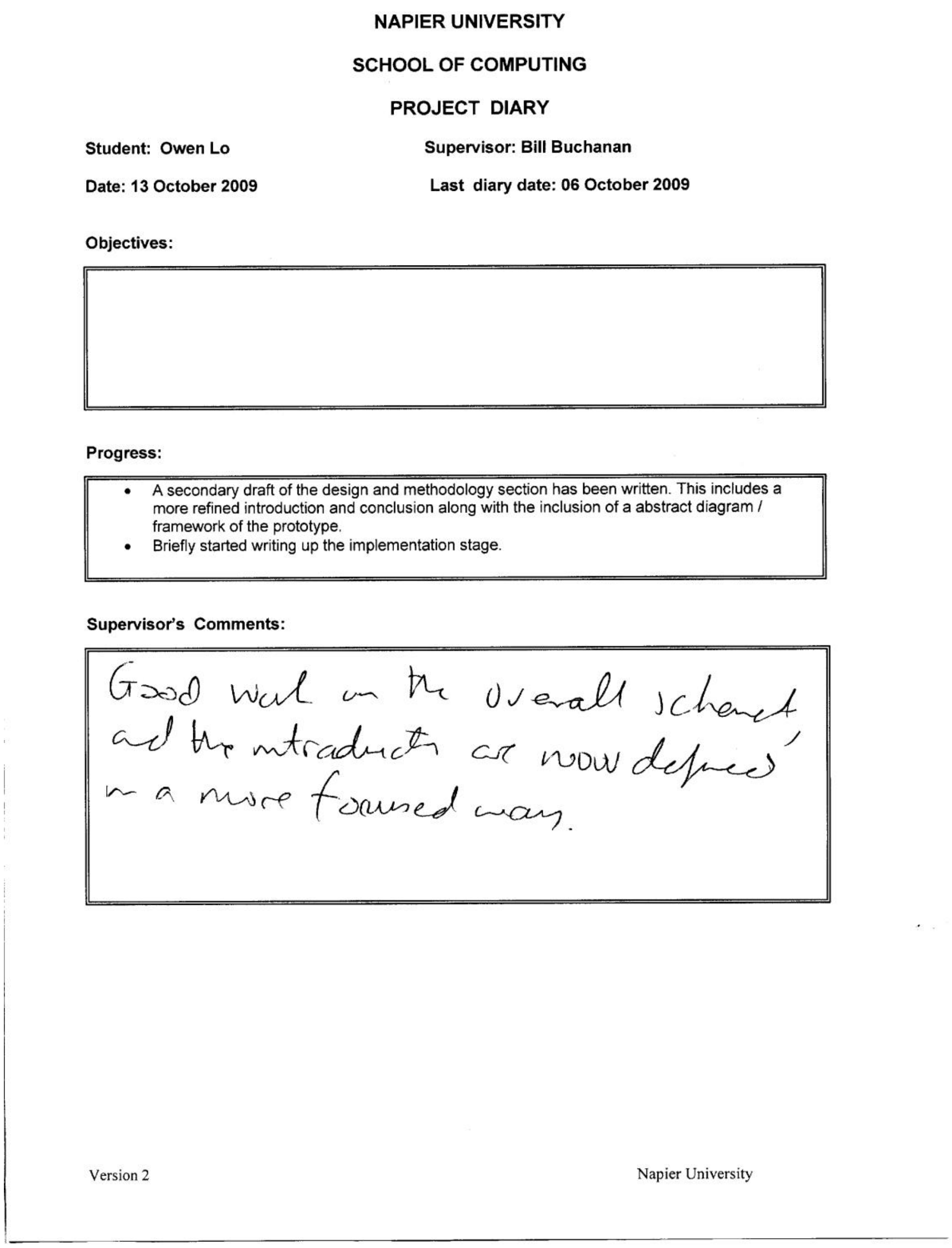
Progress:

- A draft of the Design and Methodology section is completed, requires a read over by supervisor.
- Implementation of prototype is going well. The menu for selection of attacks is in good progress. Surveillance/Probing (NMAP), Evasion (Fragroute), Insertion (Fragroute), Denial of Service (HPING3) and User Privilege Gain (Hydra) are implemented.
- Retrieval of alarms from via SSH is complete. This allows the prototype to directly copy over the logs on the system which the IDS resides and carry out calculations of metrics...
- ...which has been partly implemented. This includes the efficiency and effectiveness metric , the only refinement which needs to be made in this area is to make the way in which the data is displayed more meaningful.

Supervisor's Comments:

Good progress on the dissertation with a good narrative for the design/methodology. Have a think about and outline abstraction of the complete system, as the starting point of the Design chapter. Also focus your Introduction/Conclusions.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 13 October 2009	**Last diary date: 06 October 2009**

Objectives:

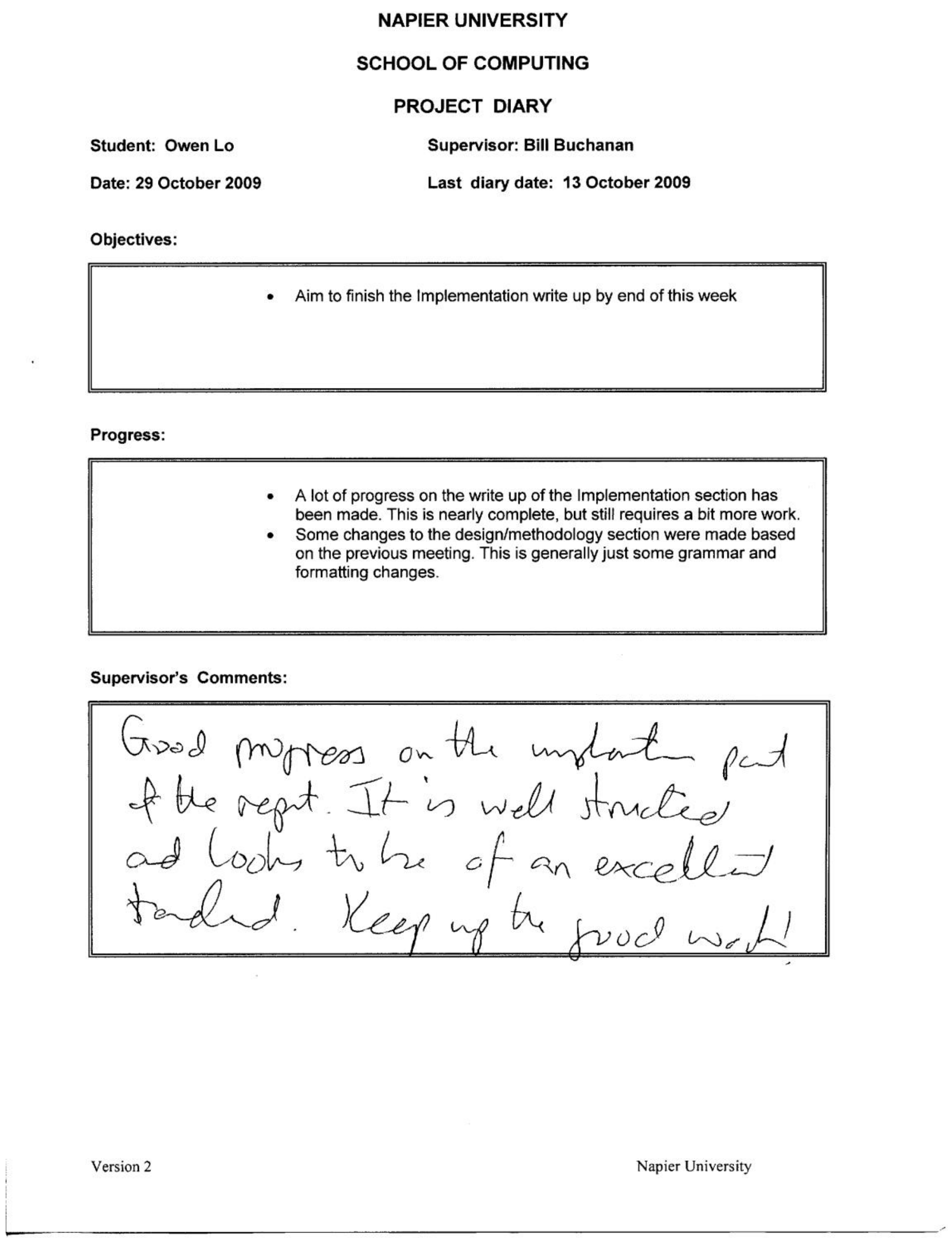
Progress:

- A secondary draft of the design and methodology section has been written. This includes a more refined introduction and conclusion along with the inclusion of a abstract diagram / framework of the prototype.
- Briefly started writing up the implementation stage.

Supervisor's Comments:

Good work on the overall scheme and the introduction are now defined in a more focused way.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 29 October 2009	**Last diary date: 13 October 2009**

Objectives:

- Aim to finish the Implementation write up by end of this week

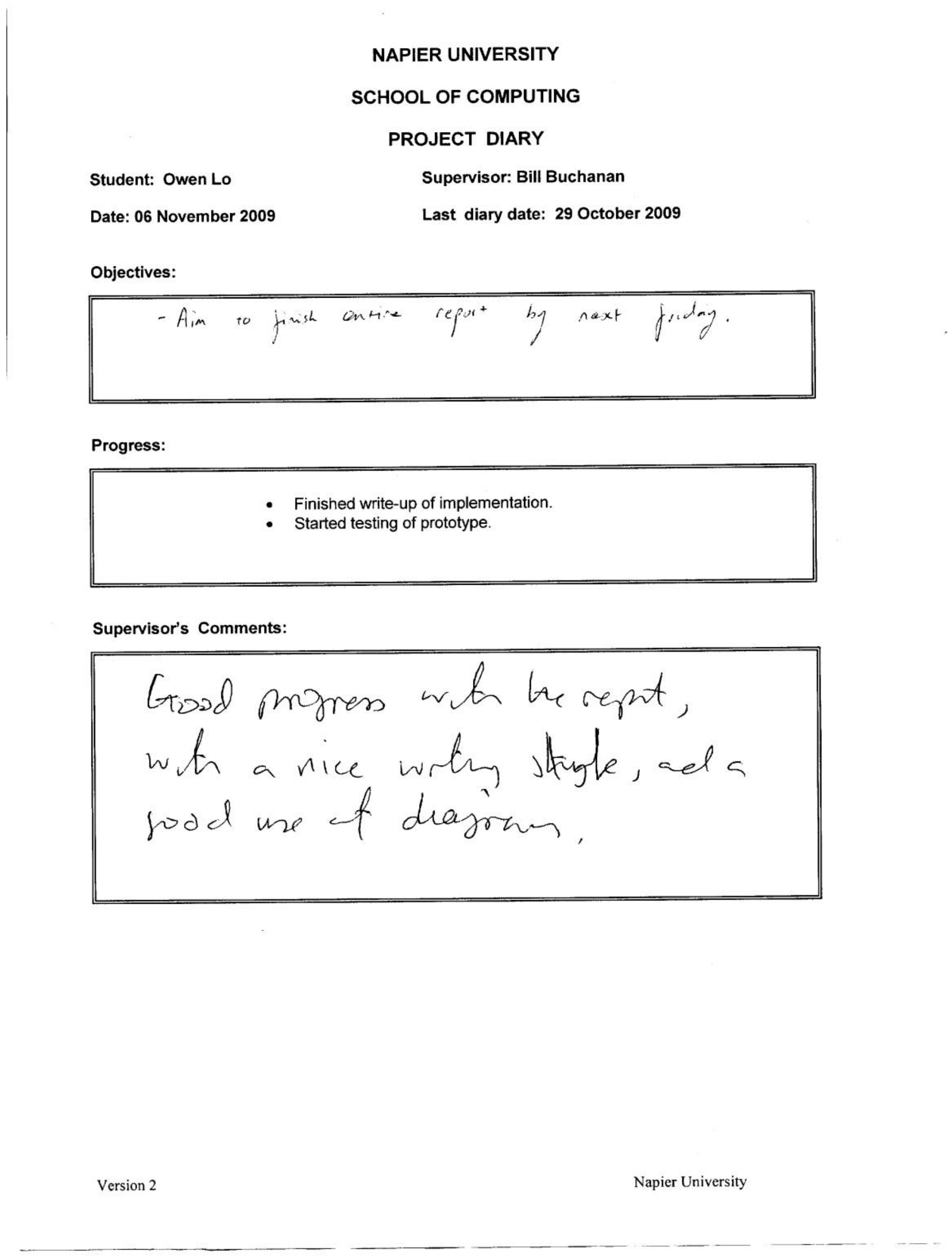
Progress:

- A lot of progress on the write up of the Implementation section has been made. This is nearly complete, but still requires a bit more work.
- Some changes to the design/methodology section were made based on the previous meeting. This is generally just some grammar and formatting changes.

Supervisor's Comments:

Good progress on the implementation part of the report. It is well structured and looks to be of an excellent standard. Keep up the good work!

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	Supervisor: Bill Buchanan
Date: 06 November 2009	Last diary date: 29 October 2009

Objectives:

- Aim to finish entire report by next friday.

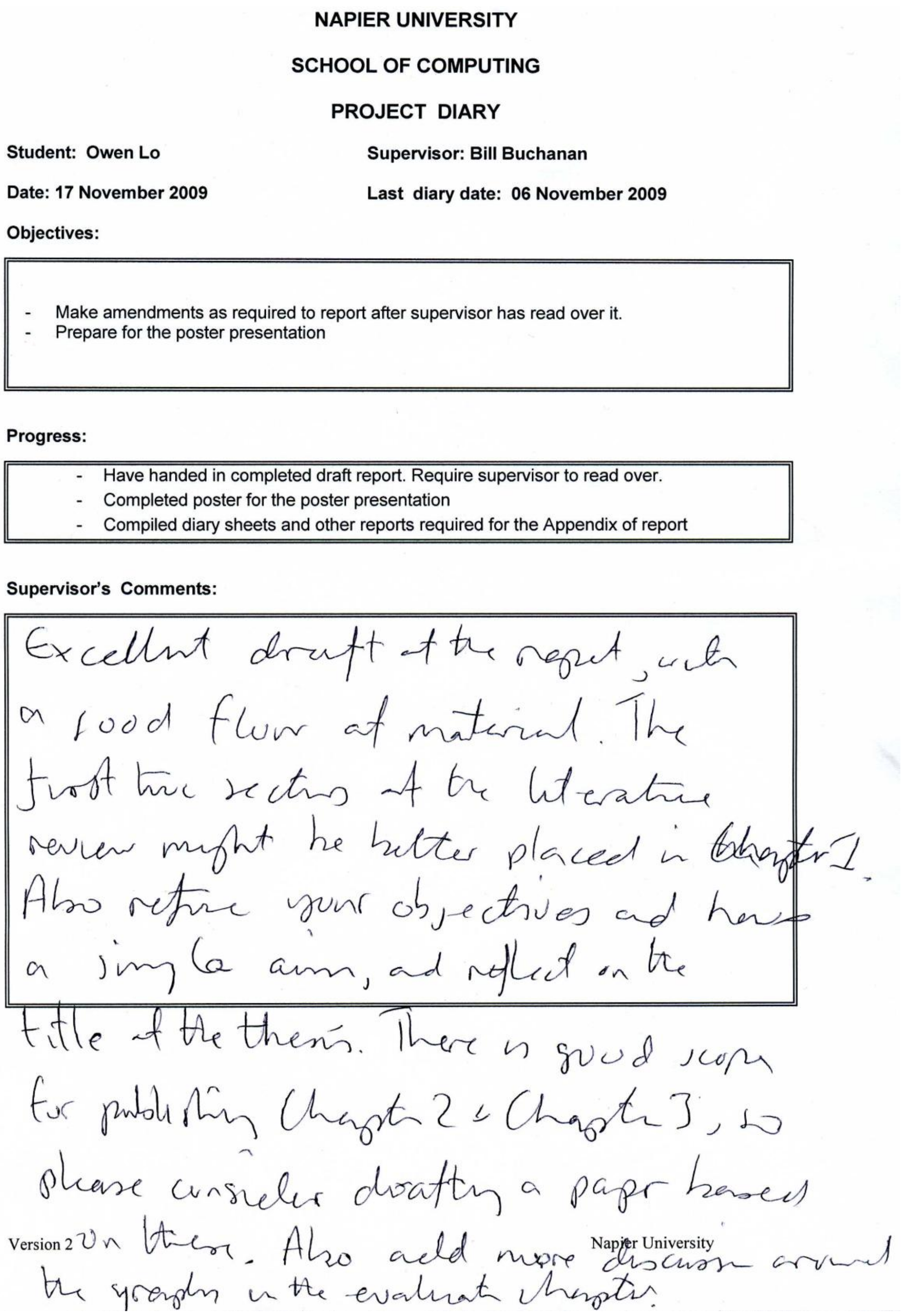
Progress:

- Finished write-up of implementation.
- Started testing of prototype.

Supervisor's Comments:

Good progress with the report, with a nice writing style, and a good use of diagrams.

Version 2 Napier University



NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Owen Lo	**Supervisor: Bill Buchanan**
Date: 17 November 2009	**Last diary date: 06 November 2009**

Objectives:

- Make amendments as required to report after supervisor has read over it.
- Prepare for the poster presentation

Progress:

- Have handed in completed draft report. Require supervisor to read over.
- Completed poster for the poster presentation
- Compiled diary sheets and other reports required for the Appendix of report

Supervisor's Comments:

Excellent draft of the report, with a good flow of material. The first two sections of the literature review might be better placed in Chapter 1. Also refine your objectives and have a single aim, and reflect on the title of the thesis. There is good scope for publishing Chapter 2 & Chapter 3, so please consider drafting a paper based on these. Also add more discussion around the graphs in the evaluation chapter.

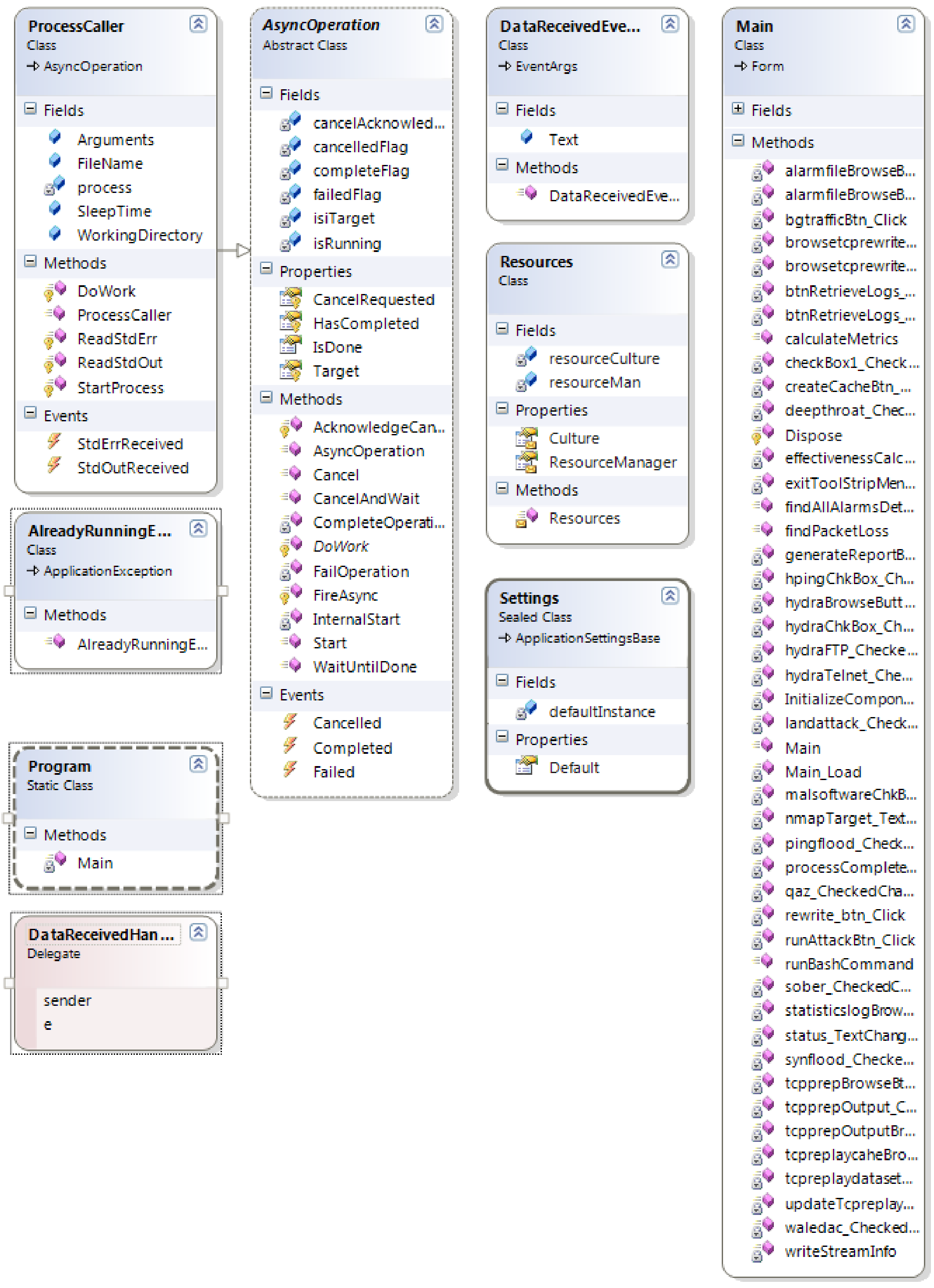
Version 2 Napier University

Appendix 5 Main.cs Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
using System.Collections;
using System.Diagnostics;
using System.Text.RegularExpressions;

namespace ProcessCaller
{
    public partial class Main : Form
    {
        public Main()
        {
            InitializeComponent();
        }

        private void Main_Load(object sender, EventArgs e)
        {

        }

        private ProcessCaller processCaller;

        public void runBashCommand(string arguments)
        {
            processCaller = new ProcessCaller(this);
            processCaller.FileName = "bash";
            processCaller.Arguments = arguments;
            processCaller.StdErrReceived += new
DataReceivedHandler(writeStreamInfo);
            processCaller.StdOutReceived += new
DataReceivedHandler(writeStreamInfo);
            processCaller.Completed += new
EventHandler(processCompletedOrCanceled);
            processCaller.Cancelled += new
EventHandler(processCompletedOrCanceled);
            this.status.Text = Environment.NewLine;
            processCaller.Start();
        }

        private void writeStreamInfo(object sender,
DataReceivedEventArgs e)
        {
            this.status.AppendText(e.Text + Environment.NewLine);
            if (status.TextLength >= status.MaxLength)
status.Clear();
        }

        private void processCompletedOrCanceled(object sender,
EventArgs e)
        {
            this.Cursor = Cursors.Default;
        }
        private void tcpprepBrowseBtn_Click(object sender, EventArgs
e)
        {
            BrowseBtnDialog.Filter = ".pcap .dump Files|*.pcap|All
Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                tcpprepInputTxtBox.Text = sFileName;
            }

            else
            {
                MessageBox.Show("Please select a dataset");
            }
        }

        private void tcpprepOutputBrowseBtn_Click(object sender,
EventArgs e)
        {


            if (OpenFolderDialog.ShowDialog() == DialogResult.OK)
            {

                String sFileName = OpenFolderDialog.SelectedPath;


                tcpprepOutputTxtBox.Text = sFileName;

            }

            else
            {

                MessageBox.Show("Output Directory is required");

            }
        }

        private void createCacheBtn_Click(object sender, EventArgs e)
        {

            string output_directory = tcpprepOutputTxtBox.Text;
            string input_file = tcpprepInputTxtBox.Text;
            this.runBashCommand("-c ' " + "tcpprep " + "-a bridge " +
"-i " + input_file + " -o " + output_directory + " ' ");
        }

        private void tcpreplaydatasetBrowse_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".pcap .dump Files|*.pcap|All
Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;


                tcpreplayDatasetInputTxtBox.Text = sFileName;

            }
        }

        private void status_TextChanged(object sender, EventArgs e)
        {

        }

        private void tcpreplaycaheBrowse_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".cache File|*.cache|All
Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                tcpreplayCacheInputTxtBox.Text = sFileName;
            }
        }

        private void bgtrafficBtn_Click(object sender, EventArgs e)
        {
            this.runBashCommand(" -c ' " + tcpreplayCommandBox.Text +
" ' ");
        }


        private void checkBox1_CheckedChanged(object sender,
EventArgs e)
        {


            if (surprobChkBox.Checked == true)
            {
                nmapGroup.Enabled = true;
            }
            else

                if (surprobChkBox.Checked == false)
                {
                    nmapGroup.Enabled = false;
                    this.nmapTarget.Text = null;
                    this.nmapScanCommandBox.Text = null;
                }

        }




        private void hpingChkBox_CheckedChanged(object sender,
EventArgs e)
        {
            if (dosChkBox.Checked == true)
            {
                hpingGroup.Enabled = true;
            }
            else
                if (dosChkBox.Checked == false)
                {
                    hpingGroup.Enabled = false;
                    this.hpingPingCommandBox.Text = "";
                    this.hpingTarget.Text = "";
                }
        }


        private void runAttackBtn_Click(object sender, EventArgs e)
        {

            if (surprobChkBox.Checked == true)
            {
                string nmap_scan = this.nmapScanCommandBox.Text;
                this.runBashCommand("-c ' " + nmap_scan + " ' ");

            }

            if (dosChkBox.Checked == true)
            {
                string hping_ping = this.hpingPingCommandBox.Text;
                string hping_SYN = this.hpingSYNCommandBox.Text;
                string hping_LAND = this.hpingLANDCommandBox.Text;
                this.runBashCommand("-c ' " + hping_ping + " ' ");
                this.runBashCommand("-c ' " + hping_SYN + " ' ");
                this.runBashCommand("-c ' " + hping_LAND + " ' ");
            }

            if (malsoftwareChkBox.Checked == true)
            {
                string qaz = this.qazCommandBox.Text;
                string sober = this.soberCommandBox.Text;
                string deepthroat = this.deepthroatCommandBox.Text;
                string waledac = this.waledacCommandBox.Text;
                this.runBashCommand("-c ' " + qaz + " ' ");
                this.runBashCommand("-c ' " + sober + " ' ");
                this.runBashCommand("-c ' " + deepthroat + " ' ");
                this.runBashCommand("-c ' " + waledac + " ' ");
            }


            if (usrGainChkBox.Checked == true)
            {
                string hydraTelnetCommand =
this.hydraTelnetCommandBox.Text;
                string hydraFTPCommand =
this.hydraFTPCommandBox.Text;
                this.runBashCommand("-c ' " + hydraTelnetCommand + "
' ");

                this.runBashCommand("-c ' " + hydraFTPCommand + " '
");
            }
        }

        private void exitToolStripMenuItem_Click(object sender,
EventArgs e)
```

```
        {

            Application.Exit();

        }



        private void btnRetrieveLogs_Click(object sender, EventArgs
e)
        {
            String username = this.sshUsername.Text;
            String target = this.sshTargetIP.Text;
            String remotedirectoryAlarm =
this.sshRemoteDirectoryAlarm.Text;
            String remotedirectoryStatistics =
this.sshRemoteDirectoryStatistics.Text;
            String filenameAlarm = this.sshFileNameAlarm.Text;
            String filenameStatistics =
this.sshFileNameStatistics.Text;
            String localdirectory = this.sshLocalDirectory.Text;
            this.runBashCommand("-c ' " + "scp " + username + "@" +
target + ":" + remotedirectoryAlarm + filenameAlarm + " " +
localdirectory + " ' ");
            this.runBashCommand("-c ' " + "scp " + username + "@" +
target + ":" + remotedirectoryStatistics + filenameStatistics + " " +
localdirectory + " ' ");
            this.runBashCommand("-c ' " + "echo Log has been
retrieved" + " ' ");

        }


        private void alarmfileBrowseBtn_Click(object sender,
EventArgs e)
        {
            string alarmFile = this.sshFileNameAlarm.Text;
            if (OpenFolderDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = OpenFolderDialog.SelectedPath;
                sshLocalDirectory.Text = sFileName;
            }
            else
            {
                MessageBox.Show("Output Directory is required");
            }
        }

        private void alarmfileBrowseBtn2_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".ids File|*.ids|All Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                alarmLog.Text = sFileName;
            }
        }
        private void statisticslogBrowserBtn_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".log|*.log| All Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                statisticsLog.Text = sFileName;
            }
        }


        private void hydraChkBox_CheckedChanged(object sender,
EventArgs e)
        {
            if (usrGainChkBox.Checked == true)
            {
                hydraGroup.Enabled = true;
            }
            else
                if (usrGainChkBox.Checked == false)
                {
                    hydraGroup.Enabled = false;
                    this.hydraTelnetCommandBox.Text = "";
                    this.hydraTarget.Text = "";
                    this.hydraUsername.Text = "";
                }
        }

        private void hydraBrowseButton_Click(object sender, EventArgs
e)
        {
            BrowseBtnDialog.Filter = ".txt |*.txt| All Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                hydraPassList.Text = sFileName;
            }

            else
            {
                MessageBox.Show("Please select a dataset");
            }
        }



private void effectivenessCalculateBtn_Click(object sender, EventArgs
e)
        {
            findAllAlarmsDetected();
            findPacketLoss();
            calculateMetrics();
        }

        public void findAllAlarmsDetected()
        {
            string statisticslog = statisticsLog.Text;
            StreamReader streamReader = new
StreamReader(statisticslog);
            string fileContent = streamReader.ReadToEnd();

            Match match = Regex.Match(fileContent, @"ALERTS: (\d+)");

            if (match.Success)
            {
                string capture = match.Groups[1].Captures[0].Value;
                allAlarms.Text = capture;
            }
        }

        public void findPacketLoss()
        {

            // opens the statistics file and reads to end
            string statisticsfile = statisticsLog.Text;
            StreamReader streamReader = new
StreamReader(statisticsfile);
            string fileContent = streamReader.ReadToEnd();

            Match match = Regex.Match(fileContent, @"Dropped:
(\d+)");

            if (match.Success)
            {
                string capture = match.Groups[1].Captures[0].Value;
                packetLoss.Text = capture;
            }

            }



        public void calculateMetrics()
        {

            string alarmfile = alarmLog.Text;
            StreamReader sr = new StreamReader(alarmfile); //make
sure this filepath exists
            string text = sr.ReadToEnd();
            sr.Close();

            Regex r1 = new Regex(@"\b" + "Portscan" + @"\b",
RegexOptions.IgnoreCase);
            MatchCollection mc1 = r1.Matches(text);

            Regex r2 = new Regex(@"\b" + "Telnet Bruteforce Attack
Detected" + @"\b", RegexOptions.IgnoreCase);
            MatchCollection mc2 = r2.Matches(text);

            Regex r3 = new Regex(@"\b" + "FTP Bruteforce Attack
Detected" + @"\b", RegexOptions.IgnoreCase);
            MatchCollection mc3 = r3.Matches(text);

            Regex r4 = new Regex(@"\b" + "QAZ Worm Detected" + @"\b",
RegexOptions.IgnoreCase);
            MatchCollection mc4 = r4.Matches(text);
            Regex r5 = new Regex(@"\b" + "Waledac Virus Detected" +
@"\b", RegexOptions.IgnoreCase);
            MatchCollection mc5 = r5.Matches(text);

            Regex r6 = new Regex(@"\b" + "Sober Virus Detected" +
@"\b", RegexOptions.IgnoreCase);
            MatchCollection mc6 = r6.Matches(text);

            Regex r7 = new Regex(@"\b" + "Deepthroat Trojan Detected"
+ @"\b", RegexOptions.IgnoreCase);
            MatchCollection mc7 = r7.Matches(text);

            Regex r8 = new Regex(@"\b" + "Ping Flood Detected" +
@"\b", RegexOptions.IgnoreCase);
            MatchCollection mc8 = r8.Matches(text);

            Regex r9 = new Regex(@"\b" + "SYN Flood Detected" +
@"\b", RegexOptions.IgnoreCase);
            MatchCollection mc9 = r9.Matches(text);

            Regex r10 = new Regex(@"\b" + "LAND Attack Detected" +
@"\b", RegexOptions.IgnoreCase);
            MatchCollection mc10 = r10.Matches(text);

            double nmap_alarm_count = mc1.Count;
            double hydraTelnet_alarm_count = mc2.Count;
            double hydraFTP_alarm_count = mc3.Count;
            double qaz_alarm_count = mc4.Count;
            double waledac_alarm_count = mc5.Count;
            double sober_alarm_count = mc6.Count;
            double deepthroat_alarm_count = mc7.Count;
            double hpingPing_alarm_count = mc8.Count;
            double hpingSyn_alarm_count = mc9.Count;
            double hpingLand_alarm_count = mc10.Count;

            // parse all alarms as a double
            double all_alarms_count =
double.Parse(this.allAlarms.Text);

            // Parse baseline input numbers into double
            int dosBaselineAlarm_int =
int.Parse(this.dosBaselineAlarms.Text);
            int malsoftwareBaselineAlarms_int =
int.Parse(this.malsoftwareBaselineAlarms.Text);
            int usrPrivBaselineAlarms_int =
int.Parse(this.usrPrivBaselineAlarms.Text);
            int survBaselineAlarms_int =
int.Parse(this.survBaselineAlarms.Text);

            // Count the total number of detected alarms
            double total_detected_alarms = nmap_alarm_count +
hydraTelnet_alarm_count + hydraFTP_alarm_count + qaz_alarm_count +
sober_alarm_count + deepthroat_alarm_count + waledac_alarm_count +
hpingLand_alarm_count + hpingSyn_alarm_count + hpingPing_alarm_count;
            truePositives.Text = total_detected_alarms.ToString();

            double total_expected_alarms = survBaselineAlarms_int +
usrPrivBaselineAlarms_int + malsoftwareBaselineAlarms_int +
dosBaselineAlarm_int;
            allPositives.Text = total_expected_alarms.ToString();
            double effectivness = total_detected_alarms /
total_expected_alarms;
                                effectivenessDouble.Text =
effectivness.ToString();

            double efficiency = total_detected_alarms /
all_alarms_count;
            efficiencyDouble.Text = efficiency.ToString();


        }


        private void generateReportBtn_Click(object sender, EventArgs
e)
        {


            String truePositives = this.truePositives.Text;
            String allAlarms = this.allAlarms.Text;
            String allPositives = this.allAlarms.Text;
            String packetLoss = this.packetLoss.Text;

            string report = "True Positives logged by IDS: " +
truePositives + " out of " + allAlarms + "Total Alarms Logged by IDS"
+ allAlarms + "";
            System.IO.StreamWriter file = new
System.IO.StreamWriter("c:\\Users\\newo\\Desktop\\IDS_Evaluation_Util
ity\\report.txt");
            file.WriteLine(report);

            file.Close();

        }

        private void rewrite_btn_Click(object sender, EventArgs e)
        {

            string dserverMAC = this.dserverMAC.Text;
            string dclientMAC = this.dclientMAC.Text;
            string sserverMAC = this.sserverMAC.Text;
            string sclientMAC = this.sclientMAC.Text;
            string serverIP = this.serverIP.Text;
            string serverSubnet = this.serverSubnet.Text;
            string clientIP = this.clientIP.Text;
            string clientSubnet = this.clientSubnet.Text;
            string tcprewritePCAP = this.tcprewritePCAP.Text;
            string tcprewritePREP = this.tcprewritePREP.Text;
            string outputdirectory =
this.tcprewriteOutputDirectory.Text;

            this.runBashCommand("-c ' " + "tcprewrite " + "--enet-
dmac=" + dserverMAC +
                                "," + dclientMAC + " --enet-smac=" +
sserverMAC + "," + sclientMAC + " -e " + serverIP + serverSubnet +
":" + clientIP + clientSubnet + " -C -c " + tcprewritePREP + " -i " +
tcprewritePCAP + " -o " + outputdirectory + " ' ");
        }
        private void hydraTelnet_CheckedChanged(object sender,
EventArgs e)
        {
            string target = this.hydraTarget.Text;
            string username = this.hydraUsername.Text;
            string passwordList = this.hydraPassList.Text;

            if (hydraTelnet.Checked == true)
            {
                this.hydraTelnetCommandBox.Text = "hydra -l " +
username + " -P " + passwordList + " " + target + " telnet";
            }
            else
                this.hydraTelnetCommandBox.Text = null;
        }

        private void hydraFTP_CheckedChanged(object sender, EventArgs
e)
        {
            string target = this.hydraTarget.Text;
            string username = this.hydraUsername.Text;
            string passwordList = this.hydraPassList.Text;
            if (hydraFTP.Checked == true)
            {
            this.hydraFTPCommandBox.Text = "hydra -l " + username + "
-P " + passwordList + " " + target + " ftp";
            }
            else
                this.hydraFTPCommandBox.Text = null;
        }

        private void pingflood_CheckedChanged(object sender,
EventArgs e)
        {
            string target = this.hpingTarget.Text;
            if (pingflood.Checked == true)
            {
                this.hpingPingCommandBox.Text = "hping3 -I eth0 -1 -i
u1000 " + target + " -c 1000";
            }
            else
                this.hpingPingCommandBox.Text = null;
        }

        private void landattack_CheckedChanged(object sender,
EventArgs e)
        {
            string target = this.hpingTarget.Text;
            if (landattack.Checked == true)
            {
                this.hpingLANDCommandBox.Text = "hping3 -S -a " +
target + " -p 21 " + target;
            }
            else
                this.hpingLANDCommandBox.Text = null;
        }

        private void synflood_CheckedChanged(object sender, EventArgs
e)
        {
            string spoof = this.hpingSpoof.Text;
            string target = this.hpingTarget.Text;
            if (synflood.Checked == true)
            {
                this.hpingSYNCommandBox.Text = "hping3 -I eth0 -a " +
spoof + " -S " + target + " -p 22 -i u1000 -c 1000";
            }
            else
                this.hpingSYNCommandBox.Text = null;
        }

        private void qaz_CheckedChanged(object sender, EventArgs e)
        {
            string target = this.malsoftwareTarget.Text;
            if (qaz.Checked == true)
            {
                this.qazCommandBox.Text = "hping3 -I eth0 -A " +
target + " -p 139 -e 'qazwsx.hsq' -c 1";
            }
            else
                this.qazCommandBox.Text = null;
        }

        private void waledac_CheckedChanged(object sender, EventArgs
e)
        {
            string target = this.malsoftwareTarget.Text;
            if (waledac.Checked == true)
            {
                this.waledacCommandBox.Text = "hping3 -I eth0 " +
target + " -p 80 -e 'X-request-kind-code:' -c 1";
            }
            else
                this.waledacCommandBox.Text = null;
        }

        private void sober_CheckedChanged(object sender, EventArgs e)
        {
            string target = this.malsoftwareTarget.Text;
            if (sober.Checked == true)
            {
                this.soberCommandBox.Text = "hping3 -I eth0 -S "
+target+ " -p 37 -e '.exe' -c 1";
            }
            else
                this.soberCommandBox.Text = null;
        }

        private void malsoftwareChkBox_CheckedChanged(object sender,
EventArgs e)
        {
            if (malsoftwareChkBox.Checked == true)
            {
                malsoftwareGroup.Enabled = true;
            }
            else
                if (malsoftwareChkBox.Checked == false)
                {
                    malsoftwareGroup.Enabled = false;
                    this.qazCommandBox.Text = "";
                    this.malsoftwareTarget.Text = "";
                }
        }

        private void deepthroat_CheckedChanged(object sender,
EventArgs e)
        {
            string target = this.malsoftwareTarget.Text;
            if (deepthroat.Checked == true)
            {
                this.deepthroatCommandBox.Text = "hping3 -I eth0 -2
"+target+ " -p 3150 -e '00' -c 1";
            }
            else
                this.deepthroatCommandBox.Text = null;
        }

        private void updateTcpreplayboxBtn_Click(object sender,
EventArgs e)
        {
            string loop = this.tcpreplayLoop.Text;
            string speed = this.tcpreplaySpeed.Text;
            string intCA = this.intCA.Text;
            string intUA = this.intUA.Text;
            string cacheLocation =
this.tcpreplayCacheInputTxtBox.Text;
            string datasetLocation =
this.tcpreplayDatasetInputTxtBox.Text;
            tcpreplayCommandBox.Text = ("tcpreplay " + "-l " + loop +
" -p " + speed + " -i " + intCA + " -I " + intUA + " -c " +
cacheLocation + " " + datasetLocation);
        }

        private void nmapTarget_TextChanged(object sender, EventArgs
e)
        {
            string target = this.nmapTarget.Text;
            nmapScanCommandBox.Text = "nmap " + target;

        }

        private void browsetcprewritePCAP_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".pcap .dump Files|*.pcap|All
Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                tcprewritePCAP.Text = sFileName;

            }
        }

        private void browsetcprewritePREP_Click(object sender,
EventArgs e)
        {
            BrowseBtnDialog.Filter = ".cache File|*.cache|All
Files|*.*";
            BrowseBtnDialog.InitialDirectory = @"C:\";
            if (BrowseBtnDialog.ShowDialog() == DialogResult.OK)
            {
                String sFileName = BrowseBtnDialog.FileName;
                tcprewritePREP.Text = sFileName;
            }
        }

        private void tcpprepOutput_Click(object sender, EventArgs e)
        {

            if (OpenFolderDialog.ShowDialog() == DialogResult.OK)
            {

                String sFileName = OpenFolderDialog.SelectedPath;


                tcprewriteOutputDirectory.Text = sFileName;

            }

            else
            {

                MessageBox.Show("Output Directory is required");

            }
        }

        private void btnRetrieveLogs_Click_1(object sender, EventArgs
e)
        {
            String username = this.sshUsername.Text;
            String target = this.sshTargetIP.Text;
            String remotedirectoryAlarm =
this.sshRemoteDirectoryAlarm.Text;
            String remotedirectoryStatistics =
this.sshRemoteDirectoryStatistics.Text;
            String filenameAlarm = this.sshFileNameAlarm.Text;
            String filenameStatistics =
this.sshFileNameStatistics.Text;
            String localdirectory = this.sshLocalDirectory.Text;
            this.runBashCommand("-c ' " + "scp " + username + "@" +
target + ":" + remotedirectoryAlarm + filenameAlarm + " " +
localdirectory + " ' ");
            this.runBashCommand("-c ' " + "scp " + username + "@" +
target + ":" + remotedirectoryStatistics + filenameStatistics + " " +
localdirectory + " ' ");
            this.runBashCommand("-c ' " + "echo Log has been
retrieved" + " ' ");
        }

    }
}
```

Appendix 6 Class Diagram of Framework


ProcessCaller
Class
AsyncOperation
Fields
Arguments
FileName
process
SleepTime
WorkingDirectory
Methods
DoWork
ProcessCaller
ReadStdErr
ReadStdOut
StartProcess
Events
StdErrReceived
StdOutReceived
AsyncOperation
Abstract Class
Fields
cancelAcknowled...
cancelledFlag
completeFlag
failedFlag
isiTarget
isRunning
Properties
CancelRequested
HasCompleted
IsDone
Target
Methods
AcknowledgeCan...
AsyncOperation
Cancel
CancelAndWait
CompleteOperati...
DoWork
FailOperation
FireAsync
InternalStart
Start
WaitUntilDone
Events
Cancelled
Completed
Failed
DataReceivedEve...
Class
EventArgs
Fields
Text
Methods
DataReceivedEve...
Resources
Class
Fields
resourceCulture
resourceMan
Properties
Culture
ResourceManager
Methods
Resources
Settings
Sealed Class
ApplicationSettingsBase
Fields
defaultInstance
Properties
Default
Main
Class
Form
Fields
Methods
alarmfileBrowseB...
alarmfileBrowseB...
bgtrafficBtn_Click
browsetcprewrite...
browsetcprewrite...
btnRetrieveLogs_...
btnRetrieveLogs_...
calculateMetrics
checkBox1_Check...
createCacheBtn_...
deepthroat_Chec...
Dispose
effectivenessCalc...
exitToolStripMen...
findAllAlarmsDet...
findPacketLoss
generateReportB...
hpingChkBox_Ch...
hydraBrowseButt...
hydraChkBox_Ch...
hydraFTP_Checke...
hydraTelnet_Che...
InitializeCompon...
landattack_Check...
Main
Main_Load
malsoftwareChkB...
nmapTarget_Text...
pingflood_Check...
processComplete...
qaz_CheckedCha...
rewrite_btn_Click
runAttackBtn_Click
runBashCommand
sober_CheckedC...
statisticslogBrow...
status_TextChang...
synflood_Checke...
tcpprepBrowseBt...
tcpprepOutput_C...
tcpprepOutputBr...
tcpreplaycaheBro...
tcpreplaydataset...
updateTcpreplay...
waledac_Checked...
writeStreamInfo
AlreadyRunningE...
Class
ApplicationException
Methods
AlreadyRunningE...
Program
Static Class
Methods
Main
DataReceivedHan...
Delegate
sender
e