Combination of Misuse and Anomaly Network Intrusion Detection Systems

Version 1.0
March 2002

Kaleton Internet
Dept. 5364 Suite 145
269/2 Soi Potisarn Moo 6
Naklua Banglamung
Chonburi 20150
Thailand

Telephone : +44 7905 939 479
Voicemail : +66 2 957 5664 ext 5364

E-mail : research@kaleton.com
Web: http://www.kaleton.com

Abstract

Intrusion detection is a relatively new addition to the field of computer security. It is concerned with software that can distinguish between legitimate users and malicious users of a computer system and make a controlled response when an attacker is detected.

This paper investigates the combination of two different approaches to intrusion detection. Signature-based systems and anomaly-based systems have both been used for several years but have not been combined together when monitoring network traffic.

The paper includes the implementation and documentation of a prototype Intrusion Detection System running on a Linux server.

1. Introduction

For several years now, society has been dependent on information technology. With the rise of the Internet and e-commerce this is truer now than ever. People rely on computer networks to provide them with news, stock prices, email and online shopping. People's credit card details, medical records and other personal information are stored on computer systems. Many companies have a web presence as an essential part of their business. The research community uses computer systems to carry out its work and share knowledge. National infrastructure components such as the power grid are controlled by computer. The integrity and availability of all these systems needs to be defended against a number of threats. Amateur hackers, rival corporations, terrorists and even foreign governments have the motive and capability to carry out sophisticated attacks against computer systems. Therefore the field of computer security has become vitally important to the safety and economic well being of society as a whole.

Every organisation using information systems must take computer security seriously. The first step is to produce a security policy defining what behaviour is and is not allowed. This policy can restrict which web sites users are allowed to visit, which files each user is allowed to access and which applications each user is allowed to use. The policy should identify which members of the organisation are responsible for security and how they should respond to different types of incident. The policy is then enforced using a variety of security measures. Access controls such as file permissions and username/password pairs may be used. Technologies such as firewalls and cryptographic protocols are also deployed in an attempt to prevent unauthorised access to systems and data.

The systems must be defended from insider misuse as well as external attackers. To provide accountability, system usage is usually logged. These logs provide evidence of misuse and can be used to identify the user(s) responsible. However, at large sites it is impractical for humans to monitor the log files due to their huge size. It is also becoming more and more difficult to spot misuse due to the increasing number and sophistication of attacks.

This has led to the field of automated intrusion detection. Rather than have system administrators constantly observing firewall logs and watching out for malicious behaviour, software exists to perform this function. The basic idea behind an Intrusion Detection System (IDS) is to monitor system usage and somehow identify behaviour that may go against the security policy. There are several approaches to this including neural networks, expert systems and statistical modelling. These will all be covered in chapter two. When malicious behaviour is identified, the system can just notify a system administrator or it can take steps to deal with the attack itself. In the latter case, the IDS could add rules to the firewall to deny access to the attacker (if external) or disable a particular user account.

This project will involve the design and implementation of a prototype Intrusion Detection System that can be used to monitor TCP network traffic to an Internet host and identify attacks. The IDS will be a hybrid system combining two different techniques in order to gain the benefits of both. The remaining sections of this report are as follows:

- Chapter 2: Overview of Intrusion Detection – This section covers past research into the field of intrusion detection. Different approaches to the problem are covered and notable papers are referenced. The ethical issues of intrusion detection are covered. Methods of attacking, subverting and defeating intrusion detection systems are detailed followed by

possible offensive uses of IDS. Ways of evaluating intrusion detection systems based on their ability to detect attacks are discussed. Finally, the likely direction of future research in the field is commented on.

- Chapter 3: System Specification and Design – A specification for combining both anomaly detection and misuse detection in a network-based IDS is proposed. All design decisions are justified and compared to their alternatives.
- Chapter 4: Attacks and System Response – The IDS is tested to observe how it responds to both legitimate use and misuse.
- Chapter 5: Evaluation and Future Work – The successes and failures of this project are discussed. Possible improvements are identified and long-term research issues are proposed.
- Chapter 6: Summary and Conclusions – Closing comments.
- Appendices – The content of the attack signature database and the source code of the IDS are given. Research papers referenced are also listed.

2. Overview of Intrusion Detection

Detecting computer break-ins and other malicious behaviour is a signal detection problem. The aim is to distinguish malicious use (signal) from legitimate use (noise). There are currently several different approaches to this problem and several different Intrusion Detection System (IDS) implementations available. Below we examine issues concerning the development, testing and operational use of intrusion detection systems.

1. Host based, Network based and Hybrid IDS

The two main types of intrusion detection system are host-based IDS and network-based IDS. Network-based IDS (NIDS) monitors traffic between hosts whereas host-based IDS monitors activity on the hosts themselves. Host-based IDS usually examines user activity and network based IDS usually examines the output of a packet sniffer. A sniffer is a program that reads raw packets off a network, usually after putting the network interface (eg ethernet card) into promiscuous mode. In promiscuous mode, the network interface will receive all traffic on the local network segment rather than just the packets addressed to itself.

An example of a host based IDS is Psionic HostSentry [22, Psionic], which is a system that performs Login Anomaly Detection (LAD). HostSentry keeps a record of login time and location for each user as well as activity during each session and uses this information to spot intruders masquerading as legitimate users. The following is from [22, Psionic]: "This tool allows administrators to spot strange login behavior and quickly respond to compromised accounts and unusual behavior. HostSentry incorporates a dynamic database and actually learns the user login behavior. Modular signatures to detect unusual events then utilize this behavior."

Another host-based system is Tripwire [23, Tripwire]. Tripwire detects changes to the file system on the host it is monitoring by creating a unique fingerprint for each file and generating an alert whenever the file's signature changes. The signature is generated by applying a hash function to a portion of the file. This system will detect intrusions when the attacker installs a 'root kit' with Trojaned versions of system commands such as 'ps' (UNIX command to list current processes). This is a common technique used by intruders to ensure they maintain access to the system and are not detected by the system administrator. Also available are Tripwire for Routers and Switches and Tripwire for Web Pages. Tripwire for Routers and Switches works as already explained but with router configuration files. Tripwire for Web Pages defends against web site defacements by detecting changes to HTML web pages.

A typical example of a network-based IDS is Snort [24, Snort]. The following is taken from the Snort documentation: "Snort is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes". Snort can defend a single machine or an entire network segment as it can put the network interface into promiscuous mode.

As described in [12, Inella], network based systems such as Snort cannot detect attacks if the traffic is encrypted (for example when secure sockets layer (SSL) is in use). This is because the NIDS requires access to the data part of the packets, not just the headers, to detect intrusions. In the future, encrypted traffic is likely to be used more and more. This is likely to become a very serious problem for network-based systems. Network-based systems also cannot be used to defend switched networks unless the IDS is incorporated into the switch itself. This is because the NIDS needs to put the network interface into promiscuous mode in order to capture all packets on the network, not just the ones addressed to the system on which the NIDS is running. Switched networks cannot be used in this way.

Some intrusion detection systems make use of both host based and network based elements. These are one type of hybrid intrusion detection system. An example of this is the Distributed Intrusion Detection System (DIDS) mentioned in [17, Axelsson]. This system has both a host monitor that performs host-based intrusion detection and a LAN monitor that analyses packets on the network.

2. Distributed IDS

A problem with having a fixed centralised host for intrusion detection analysis is that the bigger the network is, the more power is demanded of this host. This makes it impractical for large networks. Instead, [1, White and Pooch] suggests that each host run a process, called a Cooperating Security Manager (CSM), which analyses the activity on that host. The individual CSMs share information on users who are active on more than one host. Each CSM consists of five components. The Local Intrusion Detection System component detects intrusions on the host the CSM is running on. The Distributed Intrusion Detection component communicates with other CSMs on the network. The User Tracking System keeps a record of which hosts a user is logged into. The Intruder Handling System component works out the best course of action once an intrusion is

detected and the User Interface component interacts with the security officer. A "suspicion level" is produced for every user on the network indicating how likely it is that he or she is acting maliciously. This is useful, as it is not always possible to give a simple yes or no answer to the question of whether a given user is acting improperly. This kind of IDS will scale well to very large networks.

Host based, network based and distributed systems all make use of either misuse detection or anomaly detection to distinguish between malicious and legitimate use. These are now described below.

3. Misuse Detection

The misuse detection approach to intrusion detection is based on somehow defining what malicious behaviour is and then monitoring for it. This approach is very good at detecting attacks which are known but will miss new attack methods unless they are just minor variations on old attacks. There are a number of approaches to misuse detection.

[3, Garvey and Lunt] proposes creating a set of scenario models. A scenario model specifies the steps involved in a known attack in terms of user actions. This is described as an extension to IDES (Intrusion Detection Expert System) involving model-based reasoning. During operation, there is a set of ‘active models’. These are the scenario models which the IDS currently has evidence to suggest are occurring. Based on this, the ‘anticipator’ part of the system predicts what user actions should occur next. The ‘planner’ part of the system uses this expected user action to work out what will be expected to appear in the audit logs. The ‘interpreter’ part compares this to what actually happens and updates the set of active models based on this. This whole process is repeated until the evidence for a particular scenario exceeds a specified threshold, at which point an intrusion has been detected.

According to the paper, this technique will be very fast as the IDS only needs to examine the parts of the audit data which are relevant to the active scenario models, rather than examining the whole audit data. However, it seems likely that IDES must actually examine audit data relevant to all scenario models, not just the active models, in order to keep the set of active models up to date. Of course, this is still a subset of the whole audit data.

When applying misuse detection to TCP/IP network traffic, an attack signature database such as that generated by arachNIDS (Advanced Reference Archive of Current Heuristics for Network Intrusion Detection Systems) at http://whitehats.com/ids/ can be used. These databases have a small signature for each attack in the form of source and destination port, header flags and a small string present in the data part of the packet. An example of such an attack signature is:

```
alert TCP $EXTERNAL any -> $INTERNAL 27665 (msg:
"IDS525/ddos-trin00-attacker-to-master-gOrave"; flags: A+;
content: "gOrave";)
```

This signature can be used to detect network traffic that indicates the host being defended has a copy of the trin00 master on it. Trin00 is a distributed denial of service

(DDoS) tool that consists of a client, a server (master) and an agent. The master program is installed on compromised hosts, allowing the attacker to launch DoS attacks by remote control from the client program. By default, the master listens on port 27665 for TCP connections and requires the password "gOrave". These default values are what allows the above signature to work. Of course, the attacker can always recompile the master to listen on a different port and use a different password. Signatures like this can be generated for many known attacks and used by an IDS to perform misuse detection.

Signatures of attacks directed specifically at web servers are covered in [6, Zenomorph]. This paper describes a number of things to look for in the content of requests made by web clients. It covers syntax that might indicate attempts at exploiting web server and CGI vulnerabilities, such as encoding requests in hexadecimal, common commands requested by attackers and evidence of buffer overflow attempts. It is not a detailed paper on IDS, and only covers signatures for attacks on web servers.

4. Anomaly Detection

In contrast to misuse detection, anomaly detection works by building a model to represent normal system usage and then monitoring for anything that does not fit this model. This approach is good at detecting new attacks that a system using misuse detection would miss.

One approach to this is the use of expert systems. This is used in the Multics Intrusion Detection and Alerting System (MIDAS) and is described in [4, Sebring et al.]. An expert system consists of a set of facts (knowledge base) and a set of rules. Each time a user action occurs it is added to the fact base. MIDAS keeps a statistical model for each user indicating how they normally use the system. This consists of mean and standard deviation values for occurrences of different activities. If a user varies from his mean by more than some multiple of his standard deviation a fact would be asserted and added to the knowledge base.

Neural networks can be trained to recognise ‘normal’ system usage and treat everything else as anomalous. [19, Ryan et al] covers a system called Neural Network Intrusion Detector (NNID) which recognises users based on which commands are executed and how frequently within a user session. This allows the detection of an intruder who masquerades as a legitimate user.

Host based systems which apply anomaly detection to user behaviour will have problems building models for users with erratic behaviour. Users who modify their behaviour slowly over a period of time can also change their model to the point where malicious behaviour is considered ‘normal’ and will not generate an alert.

When applying anomaly detection to TCP/IP network traffic, statistics can be used to detect unusual traffic. A mean and standard deviation model representing the traffic between the hosts can be constructed by the IDS. This will show what kind of traffic is ‘average’ and how much it is allowed to differ from this by. Alternatively, these values can be set manually.

A slightly different approach to anomaly detection is proposed in [7, Sasha]. This paper proposes that a rigid definition of allowed use is specified and any activity that does not fit this definition is flagged. The author refers to this as ‘strict anomaly’ detection or ‘not use’ detection. However, this is referred to in [17, Axelsson] as specification-based intrusion detection and is said to be a variation of misuse detection rather than anomaly detection as it has no self-learning component. This approach would entirely eliminate false positives as anything outside the set of allowed actions must be of security relevance. It should also detect all current and future attacks as these too will be outside the set of allowed actions. However, [7, Sasha] does not give a detailed proposal for how this rigid definition of allowed use is to be specified.

5. Ethics of Intrusion Detection

Since intrusion detection involves monitoring people’s activity there is an issue of privacy to address. [5, Shaefer] is a short paper covering the legal and ethical issues. It is noted that a company has a right to protect its computer systems but employees and other users also have a right to privacy. It is particularly noteworthy that one of the reasons a company wants to protect its computer systems is to protect the privacy of customers whose personal data may be stored. A balance between privacy and security must be reached.

Monitoring employees can lead to low morale and resentment as they feel that they are not trusted and are constantly under suspicion. The aim is to monitor in such a way that it is for the users’ good as well as the company’s, as is the case with monitoring luggage at airports. These issues influence the decision of exactly what the IDS should monitor. For example, most people would think that monitoring how many failed login attempts they make would not invade their privacy but logging every keystroke they make would be.

6. Defeating Intrusion Detection Systems

Although not much research has been done into techniques for defeating IDS several methods are known about. [18, Ptacek and Newsham] covers three classes of attack: insertion; evasion; and denial of service. Insertion and evasion both work because the IDS is not able to predict whether any given machine it is protecting will see any given packet it has captured. Different operating systems and TCP/IP stack implementations react differently to identical traffic. Some packets will be rejected and others will not. Insertion occurs when an attack is interspersed with packets that the IDS will process but the target system will not. Evasion occurs when an attack is interspersed with packets that the IDS will reject but the target system will process. In both cases, the stream of traffic seen by the IDS is different to the stream of traffic seen by the target system. This can result in the IDS not detecting the attack.

Denial-of-service attacks are used to shut down the IDS before attacking the rest of the network. DoS attacks can exploit a software bug to crash the system or exhaust one or more resources by flooding the system with traffic. If an IDS is on the receiving end of a flood attack it may spend so long processing the flood of packets that by the time it

comes to process the real attack it is too late.

[20, Rain Forest Puppy] covers the tactics used by Whisker, a CGI vulnerability scanner, to evade detection by IDS. Most of its techniques work by changing the appearance of HTTP requests without changing their meaning. For example, a common CGI vulnerability that people scan for is the phf script that used to be installed by default with the Apache web server. This script has a vulnerability allowing people to run commands on the server as whatever user Apache is running as (sometimes root!) from their web browser. To get a listing of the /etc/passwd file to run a dictionary attack against you would type the following into your browser:

```
http://victim.com/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
```

This would run '/bin/cat /etc/passwd' on the server and the output would be displayed in your browser. This is a well-known attack and so most misuse IDS will detect it with a signature for '/cgi-bin/phf' appearing in web requests. So, Whisker sends a request for '/./cgi-bin/./phf' and goes undetected.

7. Offensive Uses of Intrusion Detection Systems

An interesting application of misuse NIDS for attack rather than defence is a technique known as passive network mapping. By monitoring traffic a NIDS can work out which services and operating systems are running not only on local computers but also on any remote networks which local users communicate with. Traditionally attackers have used active mapping for network discovery. Active mapping involves sending out packets and listening for responses to construct a model of a target network. Passive mapping differs in that it does not involve sending packets, only observing network traffic. Instead of using signatures that represent attacks, the NIDS uses signatures that represent different operating systems and services. Signatures for services can be based on standard ports and server banners. For example, seeing the string 'Microsoft IIS 5.0' in a packet originating from port 80 on a system means it is probably running Microsoft's IIS 5 web server. Signatures for operating systems can be based on the way different operating systems treat identical packets due to having different TCP/IP stack implementations. This is covered in [25, Fyodor].

Using a NIDS in this way has a couple of advantages over the traditional approach to network discovery. When using active mapping only systems which happen to be switched on at the time the network is scanned will be probed. Passive mapping will allow the discovery of systems with low uptime. Another big advantage is that passive mapping does not involve generating any traffic and is therefore invisible. The packets sent during active mapping can be blocked by firewalls and will be detected by the network being scanned.

Installing such a modified misuse NIDS on key traffic choke points would result in a large amount of information about networks being gathered invisibly prior to launching an attack. Some more information on passive techniques is available in [24, Nazario].

8. Measuring the Efficiency of an Intrusion Detection System

The efficiency of an Intrusion Detection System can be measured by the number of false positives and false negatives it produces. A false positive is produced if the IDS claims there has been an intrusion when there has not. A false negative occurs if the IDS fails to detect an intrusion.

[9, Abren] covers the calculation of the Bayesian Detection Rate (BDR) for a system. This is the probability that there has been an intrusion, given that the IDS claims there has been. The formula given is shown below :

P(Ii | Ai) = (P(Ii) P(Ai | Ii)) / (P(Ii) . P(Ai | Ii) + P(Ij) P(Ai | Ij))

Where :
Ii = Intrusive behaviour
Ij = Normal behaviour
Ai = Alarm
Aj = No alarm

Thus it can be seen that lowering the rate of false positives (and hence P(Ai | Ij)) will increase the Bayesian Detection Rate. The above formula can be used to evaluate an IDS by recording the number of false positives, false negatives and true positives over a period of time. In [9, Abren], this is used to compare a number of commercial intrusion detection systems.

In order to know when a false positive, false negative or true positive has occurred we need to generate traffic consisting of a mix of attacks and legitimate use. There are a number of scripting languages designed for simulating attacks. Two of these are NASL [15, Deraison] (Nessus Attack Scripting Language) and CASL [16, NAI Inc] (Custom Audit Scripting Language). The Nessus security scanner includes a NASL interpreter that can be used to run NASL scripts on Linux machines. NASL allows easy use of sockets allowing all sorts of probes and attacks to be implemented quickly. For example, a simple TCP port scanner can be written in a few lines. The following code is taken from [15, Deraison]:

```
start = prompt("First port to scan?");
end = prompt("Last port to scan?");
for (i = start; i<end; i=i+1) {
   soc = open_sock_tcp(i);
   if (soc) {
      display("Port ",i ," is open\n");
      close(soc);
   }
}
```

NASL also allows easy spoofing of IP addresses, mainly useful for applying denial of service (DoS) attacks.

CASL is fairly similar. The following is taken from [16, NAI Inc]: "CASL is a high-level programming language designed to write programs (often called scripts) that simulate low-level attacks or information gathering checks on networks. To write programs that

simulate an attack or information gathering check, you need to write code that constructs packets and then sends those packets to a host on a network just as an actual attack or information gathering check would. You can execute the programs you create in CASL to determine if a network is vulnerable to the attack or the information gathering check simulated by the programs".

A sample TCP stealth port scanner is given in [16, NAI Inc]. This differs from the port scanner given in NASL above in that no full connections are made to the server being probed. A TCP SYN packet is sent to each port to see if a TCP ACK is returned. The code is as follows:

```
#include "tcpip.casl"
#include "packets.casl"

for(i =1; i <1023; i =i + 1) {
   OurSYN = copy SYN;
   OurSYN.tcp_source = 10;
   OurSYN.tcp_destination = i;
   OurIP = copy TCPIP;
   OurIP.ip_source = 127.0.0.1;
   OurIP.ip_destination = 127.0.0.2;
   OurPacket = [OurIP,OurSYN ];
   ip_output(OurPacket);
   OurFilter = [ "src host ", 127.0.0.2, " and tcp src port ",i
];
   ReadPacket = ip_input(2000, OurFilter);
   if(!ReadPacket) continue;
   if(size(ReadPacket) < size(IP) + size(TCP)) continue;
   ReadIP=extract ip from ReadPacket;
   ReadTCP=extract tcp from ReadPacket;
   if(ReadTCP.tcp_ack != 1 || ReadTCP.tcp_syn != 1 ||
ReadTCP.tcp_rst == 1) continue;
   print("Port", i, "Alive");
}
```

9. The Future of Intrusion Detection

Above we have described issues concerning the development, testing and operational use of intrusion detection systems. The field of intrusion detection is still in its infancy and there are many areas that require further work. Some of the main problems that need to be addressed in the field are as follows:

1. It is currently impossible to detect misuse in encrypted network traffic. Increasingly, secure protocols such as secure shell (ssh) and secure HTTP (https) are being used. When using these protocols network traffic is encrypted to defeat the use of packet sniffers on systems between the client and server. Unfortunately, this also means that a NIDS cannot use attack signatures to detect misuse. This is because the NIDS requires access to the data part of the packets, not just the headers, to detect intrusions.

2. There is a need to make the IDS itself more resistant to attack. As the popularity and awareness of intrusion detection systems rises, attackers will concentrate on ways of either evading or disabling the IDS itself before attacking the rest of the network. [18, Ptacek and Newsham] covers such techniques in detail.

3. Currently most IDS products react to detected attacks merely by logging them or contacting the system administrator. Ideally, the IDS should be able to take the necessary actions to deal with the attack itself. This could involve terminating network connections, blocking IP addresses at the firewall, or, in a military context, launching an attack against the intruder. Presently, intrusion detection systems are not sufficiently accurate to trust them with this power. Attackers could actually use the IDS to help with the attack by tricking it into throwing specific users off the system or closing particular connections. This could be done by carrying out attacks with the source spoofed as the user to disconnect.

4. When designing host-based intrusion detection systems it has been common practice for some time to include both misuse detection and anomaly detection. This gives a system with the benefits of both approaches; it can detect both known attacks and novel attacks. However, network-based intrusion detection systems usually depend solely on misuse detection. Research should be done into applying both misuse detection and anomaly detection to network traffic and so producing a NIDS with the strengths of both approaches.

10. Chapter Summary

We have examined past work in the field of intrusion detection. We have detailed different approaches to intrusion detection and different systems that have been implemented in the past. Four areas in which current research is currently lacking have been identified. The rest of this project focuses on the design, implementation and evaluation of a solution to point four from section 2.9 above.

3. System Specification and Design

This chapter explains the aims of the project and how these are met. Justification for each technique used is given.

1. Objective

The objective of the project is to produce an application to detect both malicious and anomalous behaviour on a computer network by monitoring TCP traffic. While using both misuse detection and anomaly detection together is common practice in host-based IDS, research on combining these two methods in a network-based IDS is scarce. Network based intrusion detection systems usually depend solely on misuse detection. For this reason, it was decided to design and implement a NIDS with two attack detection engines; misuse detection and anomaly detection. This should result in a NIDS which can accurately detect and classify known attacks whilst also

detecting novel attacks.

2. Constraints

A Linux server with a permanent connection to the Internet is available for development and testing. Due to having only a small amount of time available, only a prototype system will be produced and documented. This will demonstrate the concepts involved and reveal opportunities for further research.

The system should be easy to port from one platform to another and require as little memory and processor power as possible. Flexibility and extensibility are both key requirements.

3. Choice of Programming Language

As most of the work carried out by the IDS is processing, manipulating and comparing strings it was decided to implement the system in PERL (Practical Extraction and Report Language). PERL is a high level scripting language with good support for string manipulation such as finding regular expressions and performing sorting. It allows easy execution of shell commands and I/O redirection thus allowing a PERL script to start another application and read its output. This ability is required for collecting and reading network traffic.

Another advantage of PERL is that it is interpreted rather than compiled. This removes the need for time-consuming recompilation every time a small change to the code is made. This reduces the overall development time which is important because of the small time scale of this project.

A disadvantage of using an interpreted language is that the software will not run as fast as it would if a compiled language was used. For this prototype system, an interpreted language should be sufficient and the advantage mentioned above is more important. For a final version a compiled language would be used to minimise the amount of processing power required by the NIDS. This will be particularly important when running the software on a network with a lot of traffic.

PERL is also highly portable and system independent as interpreters are available for most operating system and hardware combinations. This allows the prototype system to be tested on different systems if needed.

More details of the PERL scripting language can be found in [26, Schwartz and Christiansen].

The other programming language that was considered for use is C/C++. As this is a compiled language it has the disadvantage described above. Every time a small change is made to the source code the software would need recompiling thus increasing development time. Another problem with C/C++ is that it does not have the same high level support for string operations that are present in PERL. For example, to search for regular expressions in strings would require extra functions to be written if using C/C++.

4. Development Platform

The whole system is being developed on an Internet-connected Linux server. The prototype IDS is used to alert the owners on detection of an external attack. The system is running the Linux Mandrake 8.0 operating system and the Bastille firewall which is filtering all ports except 22 (SSH) and 80 (HTTP). All TCP traffic between the server and its clients is logged by the packet sniffer TCPdump [8, Bejtlich]. This includes packets addressed to all ports even though the

operating system will drop those not addressed to 22 or 80. This way the NIDS will not need any changes if the firewall rules are altered at some future date. Ports 22 and 80 are unfiltered as the server is running Secure Shell (SSH) and the Apache web server.

5. System Architecture

Figure 1 shows the architecture of the NIDS. TCPDump was chosen as the packet sniffer because it is the most functional sniffer available free of charge.

When the NIDS is first run it starts up TCPDump and redirects its output to STDIN (standard input). The output from the packet sniffer is constantly read by the NIDS which processes it and reads it into a data structure (Figure 2). This data structure is accessed by both the misuse detection engine and the anomaly detection engine for analysis. The data structure only stores the last one hundred packets and therefore requires a small and constant amount of memory.

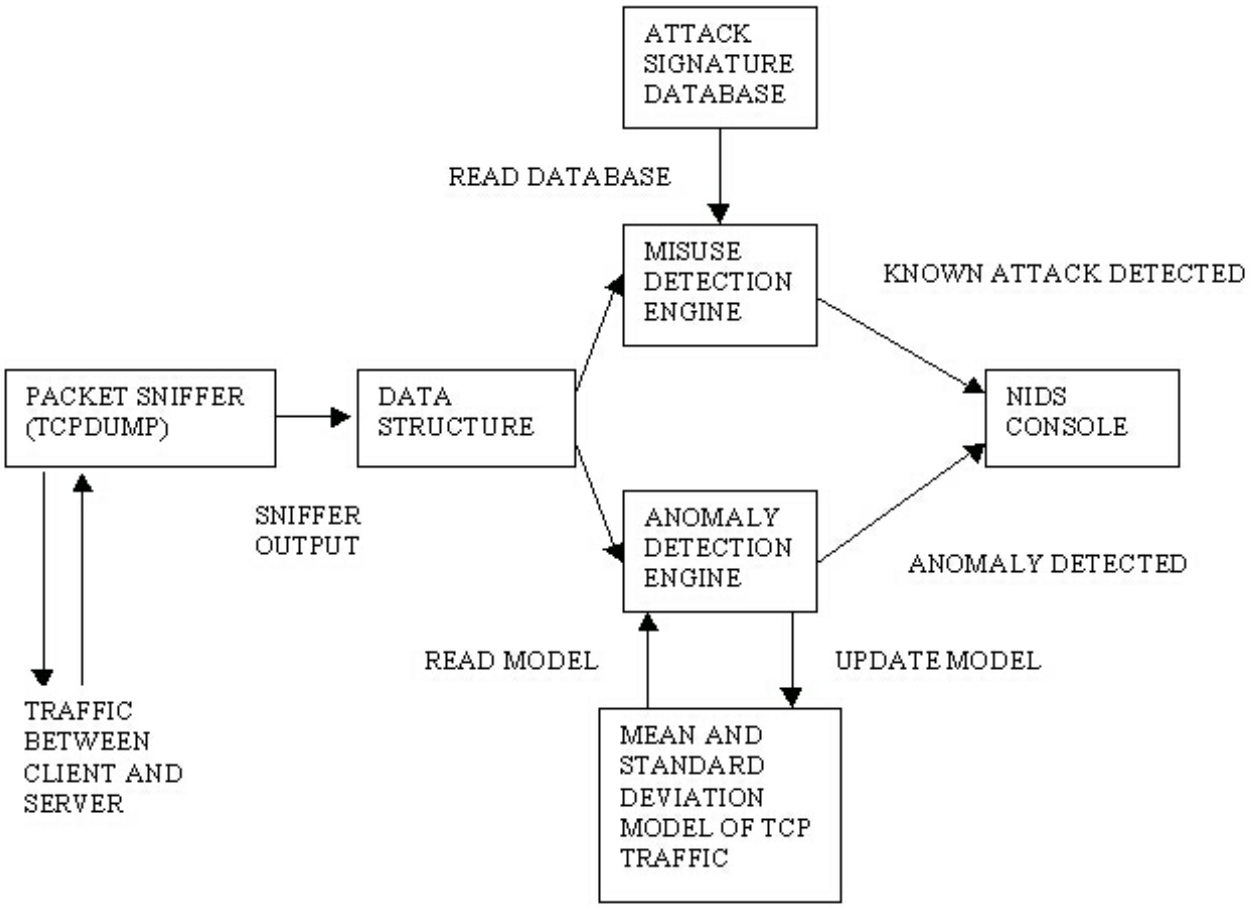

Figure 1: NIDS Architecture

1. Misuse Detection Engine

The misuse detection engine makes use of an attack signature database adapted from that generated by [21, arachNIDS]. An original attack signature database could have been created by running a vulnerability scanner such as Nessus against a system running a packet sniffer. Unique attack signatures could have been extracted from the output of the sniffer. However, creating a signature database in this way is outside the scope of this project. The database produced from [21, arachNIDS] has a small signature for each attack in the form of source and destination port, header flags and a small string present in the data part of the packet. This is used by the NIDS to detect attacks on a packet by packet basis. An advantage of this is that the implementation is relatively straightforward and also attacks are detected almost instantly. A disadvantage is that attacks broken over two or more packets using fragmentation will be missed. Commercial standard NIDS operate on streams rather than individual packets as they reassemble fragmented packets. Implementing this was outside the scope of the prototype system.

The engine compares all the signatures in the database to the sniffer data, essentially comparing strings. This part of the NIDS is very similar to existing systems such as Snort. It will efficiently detect a wide range of known attacks such as attempts to exploit known software vulnerabilities. If a match is found a message reflecting this is sent to the console.

	Packet n	Packet n+1
Time		
Source IP Address		
Destination IP Address		
Source Port		
Destination Port		
SYN		
ACK		
RST		
FIN		
URG		
PSH		
Data Portion Of Packet		

Figure 2: Data Structure for Storing Packets Internally

2. Anomaly Detection Engine

The job of the anomaly detection engine is to build and maintain a statistical model representing the traffic between the server and its clients. The model consists of mean and standard deviation values for occurrences per hundred packets of many different properties

of the traffic. This just requires keeping counters and doing some calculations every hundred packets. Therefore it is less processor-intensive than other approaches such as neural networks. When traffic is detected which does not fit in with the model (i.e. in one hundred packets the recorded number of occurrences for one or more traffic property falls outside mean ± two standard deviations) a message is sent to the console indicating that an anomaly has been detected.

An example of this working is the detection of a SYN flood attack. SYN flooding is an old and once very popular denial of service (DoS) attack. When System1 establishes a TCP connection to a given port on System2 the first part is known as the three-way handshake. It consists of System1 sending a packet to System2 with the SYN (synchronise) flag set. System2 then replies to System1 by sending a packet with both the SYN and the ACK (acknowledge) flags set. At this point System1 sends another packet to System2 that also has the SYN and ACK flags set and the connection is now established. If for some reason the final part of that interaction failed to happen, System2 would wait usually for several minutes before aborting. For this period the connection is in the WAIT state. It is neither established or not. Most operating systems can only queue a small number (eg ten) of connections in the WAIT state. Normally this would be fine as the three-way handshake is fast and the queue would never need to hold ten connections in this state at once. If, however, an attacker sends several packets with the SYN flag set and spoofs the source IP address to be that of an unreachable system then System1 will never send the final packet of the three-way handshake (because System1 does not really exist). System2's connection queue will be full and that port on System2 will no longer be reachable. As keeping the queue full only requires the attacker to send a very small number of forged packets to System2 every few minutes it is perfectly possible to shut down several servers that have T3 lines using only a 14.4 Kbps modem. However, during a SYN flood the number of received packets with the SYN flag set would be greater than the mean value + twice the standard deviation value as appears in the statistical model of normal traffic and so this would be flagged by the anomaly detection engine.

The anomaly detection part of the IDS is very good at detecting any attack which involves sending a lot of packets. It is also capable of detecting novel malicious behaviour that is missed by the misuse detection engine. The attributes measured for the statistical model are as follows:

- A frequency histogram of the occurrences of different source ports is kept for both incoming and outgoing packets.
- A frequency histogram is also kept for occurrences of different destination ports for both incoming and outgoing packets.
- For each of the TCP flags (URG, ACK, PSH, RST, SYN and FIN) a count is kept of how many incoming and outgoing packets are seen with that flag set.
- The time taken, in seconds, for one hundred packets to be logged is monitored.

Thus we can calculate a mean and a standard deviation for occurrences in both incoming and outgoing packets of each possible source port, each possible destination port and each TCP flag per hundred packets. A mean and a standard deviation value are also calculated for the time taken to observe one hundred packets.

After the number of occurrences of these properties has been observed for a hundred packets, whether or not an anomaly has occurred, the mean and standard deviation values are recalculated as follows:

- New average = (((old average x e^-λ) + occurrences in last hundred) / (e^-λ + 1))
- New standard deviation = (((old standard deviation x e^-λ) + (new average - occurrences in last hundred)) / (e^-λ + 1))

where λ is a weighting constant used to fix how much old traffic counts for compared to the last hundred packets. The last hundred packets stored in memory can then be overwritten as the cycle starts again. Every time the average and standard deviation values are recalculated they are all written to a file called 'model'. If the IDS is for some reason shut down, when it is restarted it will read in the model file and carry on from where it left off rather than having to start building the statistical model again from scratch.

An example of this process is as follows. At a given point in time, the average number of TCP packets per hundred with the ACK flag set is 57. The standard deviation is 12. The IDS monitors traffic for another hundred packets and observes 52 TCP packets with the ACK flag set. As 57-52=5 is less than 2x12=24 no anomaly has occurred. The average is now recalculated as (((57 x e^-λ) + 52) / (e^-λ + 1)). The new standard deviation is (((12 x e^-λ) + ((((57 x e^-λ) + 52) / (e^-λ + 1)) - 52)) / (e^-λ + 1)).

3. NIDS Console

Originally, the intention was to have the NIDS log all attacks to a file that would then be presented to the user in an HTML page generated by a CGI script. This would require the user to login to an interface whenever he or she wanted to check for recent malicious activity. It was decided that it would be far better to have the NIDS get in touch with the user than vice versa. This allows the user to make a much more timely response to intrusions. The best way to do this is to have the NIDS send an email whenever it detects an attack.

The console part of the IDS uses the UNIX command sendmail to notify the system administrator by email whenever it is alerted by the misuse or anomaly detection engines that an attack (or potential attack) has been detected. When the misuse detection engine alerts the console, the originating IP address and the kind of attack will be included in the resulting email. For example, when the misuse detection engine detects an ftp probe an email such as the following results:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine

Attacker's IP: 212.69.227.182
Type of attack: Ftp-probe
```

When the anomaly detection engine alerts the console the email will just contain information about whichever property of the traffic is outside its normal bounds. For example, if the number of incoming packets with the SYN flag set and the number of outgoing packets with the FIN flag set are both anomalous, an email such as this would be sent to the system administrator:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Anomaly Detection Engine

For Incoming Packets With The SYN Flag Set,
Average is 5.35415029597069,
Allowed Deviation is 1.65944673996082,
and Counted is 61

For Outgoing Packets With The FIN Flag Set,
Average is 3.8806840603014,
Allowed Deviation is 0.740674554717149,
and Counted is 0

For the 100 packets this email refers to, the following IP
addresses were communicating with the server:

212.69.227.182 with 100 packets
```

One potential problem with this method of processing intrusions is that the attacker can use it to launch a denial-of-service attack against the system administrator. For example, every time someone tries to connect to port 21 on the server the sys admin receives an email. If a malicious user writes a simple script which sits in an infinite loop repeatedly trying to telnet to port 21 the system administrator's email account will be flooded with alerts from the NIDS. This is a problem in itself but it is also possible that the attacker does this until the system administrator's email account is full and then launches a second attack, this time against the network. Since the NIDS cannot email the sys admin (whose email account is now full) he or she may never become aware of the secondary attack.

A solution might be to generate only one alert when multiple instances of the same attack are detected from the same source in a short period of time. Even so, the same problem could occur as the result of a distributed attack or someone trying a long list of attacks against the server. The latter is common when using vulnerability scanners such as Nessus to find security holes in a system.

6. Chapter Summary

The prototype intrusion detection system described above was implemented in approximately 1000 lines of PERL. The email address of the system administrator to send the alerts to is specified internally at the top of the script. So is the value of lambda to be used by the anomaly detection engine. The software is then started by simply issuing the following command:

```
[root@server]# perl ids.pl &
```

This starts the software as a background process that will continue running even after the user has logged out.

We will now examine the system's response to a range of malicious activities.

4. Attacks and System Response

To evaluate the success of the prototype NIDS an attempt to hack the development server was simulated and the NIDS' responses were observed throughout. The attacks that made up the stages of the hacking attempt were all launched from a PC running Linux Mandrake 7.1 and using a dial-up connection to the Internet.

The attacks used were chosen due to their popularity amongst the cracker community. The Computer Emergency Response Team (CERT) keeps track of current activity by analysing incident reports and maintains a list of the current most common malicious activity on their web site at http://www.cert.org/. A good idea of which attacks against web servers occur most often was also gained by monitoring the Apache web server logs on the development server.

The purpose and technical details of each chosen attack are explained followed by the method used by the NIDS to detect it. This chapter demonstrates how the prototype system reacts to some of the most common attacks threatening computer systems today.

1. Selection of Attacks

I wanted to demonstrate the entire sequence of actions that make up a typical attempt at attacking a remote computer system. The first stage of an attack is usually information gathering. This includes port scans to determine which services are running and probes for specific services and vulnerabilities. After the initial information gathering stage is complete the attacker will usually attempt to exploit a known vulnerability in the server software discovered on the system to gain unauthorised access. If all attempts to gain access fail, the frustrated attacker may well decide to launch a denial of service attack as a last resort. For each of these three stages, we chose one or more attacks. Each chosen attack is now detailed along with its reasons for being chosen.

Information gathering:

- FIN scan – This is a popular form of stealth scanning. As no connection is actually established to the ports being scanned it goes undetected by some intrusion detection systems.
- FTP probe – As a commonly running service with known weaknesses it is very common to check for the existence of an FTP server.

- Web server probe – Many vulnerabilities have been found in web server software (particularly that written by Microsoft). As a common service the presence of a web server is often checked for.
- Microsoft FPSE probe – This is a very insecure piece of software that has resulted in many defaced web sites and is commonly checked for.
- Telnet probe – Telnet is also a very common service with known security issues and so often checked for.
- SOCKS proxy server probe – This service is popular with attackers who wish to hide their identities when attacking more systems.

Exploit a known vulnerability:

- MS IIS 5.0 .printer buffer overflow – This vulnerability allows a remote user to execute any desired command and affects a lot of systems being used to host web sites. Very many attempts at exploiting this vulnerability were seen in the Apache logs on the development server, probably due to automated propagation techniques being used by current Internet worms such as Code Red.

Launch a denial-of-service attack:

- SYN flood – This Denial-of-Service attack exploits an inherent weakness in the TCP protocol to shut down one or more ports on a server. It was used to great effect in September 1996 when the PANIX ISP in New York was shut down for over a week by an attacker who remains unidentified. Although people are increasingly turning to distributed attacks and the SYN flood can now be defended against it is still used from time to time. The difficulty of legally demonstrating a distributed DoS attack (legally acquired access to many Internet hosts would be needed) also led to the choice of this method of attack.

2. Information Gathering

To begin with, we are assumed to know nothing about the target server. We do not know which ports are open, which are closed and which are filtered. We do not know the operating system being used or which services are enabled. Before gaining access to the system we need more knowledge about it.

1. FIN Port Scan

A port scan is a technique for determining which ports on a target system are listening and hence which services are likely to be running. This is an important first stage to attacking a system as it may reveal that servers with known remotely exploitable vulnerabilities are running. It usually consists of sending a packet to each port and seeing how it responds. During a FIN port scan a packet with the FIN flag set is sent to each port on the target system. Closed ports respond with a RST packet whereas open (and, unfortunately, firewalled) ports do not reply. As all ports on the development system are either open or firewalled this particular type of port scan will not reveal any useful information in this case.

To carry out this attack we used the Linux tool nmap written by Fyodor. Nmap is the most popular and fully featured network discovery tool available for free. It can carry out several different types of port scan, remote OS detection [25, Fyodor] and produce decoy scans to make tracing the source very difficult. The following command was executed:

```
[root@titan]# nmap –sF server
```

Carrying out a full scan would have been time consuming and unnecessary as the misuse detection engine operates on a packet by packet basis and the anomaly detection engine will detect anomalies every hundred packets. The port scanner was allowed to run for a couple of minutes and then it was terminated. The following alert was then received from the NIDS (edited for brevity):

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Anomaly Detection Engine

For Incoming Packets To Destination Port 81, Average is 0,
Allowed Deviation is 0, and Counted is 2

For Incoming Packets To Destination Port 94, Average is 0,
Allowed Deviation is 0, and Counted is 2

For Incoming Packets To Destination Port 101, Average is 0,
Allowed Deviation is 0, and Counted is 2

For Incoming Packets To Destination Port 165, Average is 0,
Allowed Deviation is 0, and Counted is 2

For Incoming Packets To Destination Port 166, Average is 0,
Allowed Deviation is 0, and Counted is 2

For Incoming Packets From Source Port 55043, Average is
12.4491850432782, Allowed Deviation is 4.70007463118913, and
Counted is 45

For Incoming Packets From Source Port 55044, Average is
11.2042665389504, Allowed Deviation is 4.23006716807021, and
Counted is 52

For Incoming Packets With The SYN Flag Set, Average is
40.9284811398537, Allowed Deviation is 13.088693875117, and
Counted is 1

For Incoming Packets With The FIN Flag Set, Average is
26.3003823738576, Allowed Deviation is 7.6974903206533, and
Counted is 97

For time taken for 100 packets,
Average is 200.37386327306138 seconds,
Standard Deviation is 17.95984673992142 seconds,
```

```
and Observed is 92.669449365967 34 seconds

For the 100 packets this email refers to, the following IP
addresses were communicating with the server:
212.69.227.182 with 100 packets
```

This shows that the misuse detection engine did not detect the port scan but the scan resulted in anomalous traffic being detected by the anomaly detection engine. Packets addressed to a lot of ports that have never received packets before are seen. Almost all these packets have the FIN flag set and almost none of them have the SYN flag set. An increase in the volume of traffic is seen and all these packets have come from one IP address. This would lead a competent security officer to suspect that a FIN scan is in progress from `212.69.227.182.`

As the FIN scan did not produce any information to help with our attack we assume that the closed ports are all filtered by a firewall. We move on to checking for specific services and vulnerabilities.

2. FTP Probe

Whenever an incoming packet is addressed to port 21 and has the SYN flag set it is classified as an FTP probe by the misuse detection engine. The server is not running the FTP service and therefore any connection attempts to port 21 are assumed to be somebody with hostile intent checking for the existence of this service prior to mounting an attack. Many vulnerabilities have been found in commonly used versions of FTP servers (especially WU (Washington University) FTP). Known vulnerabilities have included buffer overflows (buffer overflows will be covered in section 4.3.1) and problems associated with allowing anonymous access. Even if the particular version of the FTP server found to be running has no known vulnerabilities a brute force password attack can be launched. It is therefore very common for an attacker to check for the existence of an FTP server on a target system.

To test for the existence of an FTP daemon the standard Linux telnet client was used to attempt to connect to port 21:

```
[root@titan]# telnet server 21
Connection refused by server
```

As FTP is not running (and port 21 is filtered by the firewall anyway) the connection was refused by the server. On the development server, FTP (and also telnet) has been replaced by the SSH (Secure Shell) service which listens on port 22. SSH allows encrypted file transfers and shell sessions thus reducing the threat to security posed by sniffers.

After attempting to telnet to port 21 the following email was received from the NIDS:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine
```

```
Attacker's IP: 212.69.227.182
Type of attack: Ftp-probe
```

This shows that the single-packet probe did not cause anomalous traffic but was detected by the misuse detection engine.

3. Web Server Probe

To find out if a web server is running on the system the following was typed into a web browser:

```
http://server/
```

This revealed a web page indicating that a web server is indeed running. As accessing the web server is perfectly legitimate this was not flagged by the misuse detection engine or classed as anomalous by the anomaly detection engine. We move on to checking for a specific vulnerability related to web server software.

4. Microsoft FrontPage Server Extensions Probe

Whenever an incoming packet is addressed to port 80 and contains the string _vti_pvt in its data portion it is classified as a Microsoft FrontPage Server Extensions probe by the misuse detection engine. A default installation of MS FPSE has world-readable password files in a folder called /_vti_pvt/ which can be accessed with a web browser. If the password files are downloaded by an attacker they can often be cracked with a dictionary attack program such as the popular 'John the Ripper'. A dictionary attack is a way of finding plain-text passwords from an encrypted password file. Every word in a dictionary is encrypted and compared to each encrypted string in the password file. A match in encrypted strings indicates that a password has been discovered. This is often very successful as most people choose passwords based on real words. Large word lists for this purpose can be downloaded off various web sites. The compromised accounts can then be used to alter the web site or access restricted areas using the MS FrontPage client application. This is a favourite technique of novice web defacers as it is easy and a list of vulnerable sites can be obtained by simply going to the Altavista search engine and typing:

```
link:/_vti_pvt/services.pwd
```

This will return a list of sites spidered by Altavista that are running MS FPSE and have a world-readable password file.

To test for the existence of MS FPSE the following was typed into a web browser:

```
http://server/_vti_pvt/
```

As Microsoft FrontPage Server Extensions is not installed and hence there is no /_vti_pvt folder the connection resulted in a 404 File Not Found error. The following email was then received from the NIDS:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine

Attacker's IP: 212.69.227.182
Type of attack: FrontPage-probe
```

5. Telnet Probe

Whenever an incoming packet is addressed to port 23 and has the SYN flag set it is classified as a telnet probe by the misuse detection engine. The server is not running the telnet service and therefore any connection attempts to port 23 are assumed to be somebody with hostile intent checking for the existence of this service prior to mounting an attack. Telnet servers are vulnerable to brute force attacks (trying lots of passwords for different usernames) if any users have weak passwords.

To test for the presence of a telnet daemon the standard Linux telnet client was used to attempt to connect to port 23:

```
[root@titan]# telnet server
Connection refused by server
```

As telnet is not running (and port 23 is filtered by the firewall anyway) the connection was refused by the server. The following email was then received from the NIDS:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine

Attacker's IP: 212.69.227.182
Type of attack: Telnet-probe
```

6. SOCKS Proxy Probe

A common technique used by attackers to obfuscate their location is to attack through one or more anonymous SOCKS proxies. An anonymous SOCKS proxy will forward TCP, UDP and ICMP packets from a client to a server and then forward the server's response back to the client. The server will only log the IP address of the SOCKS proxy, not the IP address of the real client. Many such proxies exist on the Internet. Often they have been set up to allow Internet access from a LAN via only one machine. Unfortunately, people do not realise that their proxies are accepting external connections as well as connections from the LAN. It is common practice for people in the cracker community to scan large ranges of IP addresses for proxies and lists of them are published on various web sites.

By default, SOCKS proxies run on port 1080 and so whenever an incoming packet is addressed to that port and has the SYN flag set it is classified as a SOCKS probe by the misuse detection engine. The server is not running the SOCKS service and therefore any connection attempts to port 1080 are assumed to be somebody with hostile intent checking for the existence of this service prior to mounting an attack.

To probe for the presence of a SOCKS proxy the standard Linux telnet client was used to attempt to connect to port 1080:

```
[root@titan]# telnet server 1080
Connection refused by server
```

As SOCKS is not running (and port 1080 is filtered by the firewall anyway) the connection was refused by the server. The following email was then received from the NIDS:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine

Attacker's IP: 212.69.227.182
Type of attack: socks-probe
```

3. Exploiting a Known Vulnerability

As the information gathering stage revealed the presence of a web server, we try a common web server attack.

1. Microsoft IIS 5.0 .printer Buffer Overflow

Buffer overflow attacks first became really popular in November 1996 when Phrack Magazine (http://www.phrack.com/) published an article called "Smashing the Stack for Fun and Profit". This article explained the technical details of exploiting buffer overflow conditions to gain unauthorised access to systems. Buffer overflows occur when a program attempts to write more data to a buffer (for example an array) than it can contain. The memory after the end of the buffer is overwritten by the data.

An example of this is if some user input is written to an array of type `char[]` by the `strcpy()` function in a C program. The function does not bother to check the size of the array against the size of the data being written to it. If the array is 10 bytes long but we give 15 bytes of input, the 5 bytes of memory after the array will be overwritten with the last 5 bytes of the data we supplied. An example of a situation where this might occur is when a program prompts a user for a username and password. The programmer might have specified an array of 20 bytes to read the username into but someone can give a string longer than this at the login prompt.

The reason this can be useful to an attacker is that during a function call the arguments are placed on the execution stack along with the return address. If an argument is bigger than expected its last few bytes can overwrite the return address. Hence an attacker can cause

execution of the program to skip to a chosen address in memory. The new return address should be chosen to be the address of the beginning of the buffer. The data at the beginning of the buffer should be some machine code that the attacker wants to execute. When the function call finishes, instead of execution returning to the point just after where the function call was made it starts executing the data originally passed in as an argument. These instructions are usually chosen so as to send a command shell back to the attacker.

Since 1996, many buffer overflow vulnerabilities have been found in many pieces of software and exploit code made public. In May 2001 the eEye Security Group [27, eEye] found a buffer overflow vulnerability in Microsoft's IIS 5.0 web server. This allows remote attackers to run commands on vulnerable Windows machines with SYSTEM privileges (similar to root on UNIX machines). This vulnerability affected a huge number of web sites on the Internet and proved very popular with the cracker community. The Code Red worm used this as its method of propagation, penetrating approximately 250,000 systems in 24 hours.

To attempt to exploit this vulnerability a NASL script that was written by John Lampe and included with Nessus was used. The main part of this script is shown below:

```
port = get_kb_item("Services/www");
if(!port) port = 80;
if(get_port_state(port)) {
   soc = open_sock_tcp(port);
   if(!soc) exit(0);
   req = http_get(item:"/", port:port);
   send(socket:soc, data:req);
   r = recv_line(socket:soc, length:4096);
   close(soc);
   if(!r)exit(0);
   mystring = string("GET /NULL.printer HTTP/1.1\r\n");
   mystring = string(mystring, "Host: ",crap(420),"\r\n\r\n");
   mystring2 = string ("GET / HTTP/1.0\r\n\r\n");
   soc = open_sock_tcp(port);
   if(!soc) {
      exit(0);
   } else {
      send(socket:soc, data:mystring);
      close(soc);
      soc2 = open_sock_tcp(port);
      send(socket:soc2, data:mystring2);
      incoming = recv(socket:soc2, length:1024);
      if(!incoming){
      security_hole(port);
      exit(0);
   }
}
```

This script first attempts to open a socket to port 80 on the target system. If this is successful it then sends the following HTTP request:

```
GET /NULL.prınter HTTP/1.1
Host: [420 random characters go here]
```

If the host is vulnerable this will crash the web server. The script then sends the following HTTP request:

```
GET / HTTP/1.1
```

If this does not elicit a response, the web server is assumed to have crashed and therefore is vulnerable to the buffer overflow. The Windows operating system will restart the web server after a few minutes at which point it can be used to gain remote access to the system.

To run the script it was loaded into the Nessus vulnerability scanner. This then executed the script with the development server as the target system. As the development server is running the Apache web server Nessus reported back that the target was not vulnerable and so could not be exploited in this way. The following alert was then received from the NIDS:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Misuse Detection Engine

Attacker's IP: 212.69.227.182
Type of attack: iis5-printer-overflow
```

This shows that our attempted exploit was detected by the misuse detection engine. The system detected this attack by checking for incoming packets addressed to port 80 with the string '.printer' appearing in the data portion of the packet.

4. Denial-of-Service Attack

As all attempts to gain access to the system have failed, a typical move would be to launch a denial of service attack.

1. SYN Flooding

The SYN flood denial-of-service attack was described in section 3.5.2 of this report. It is based on sending a flood of forged TCP packets, with the SYN flag set, to a specific port on a target system. The source IP of the packets is spoofed to be that of a non-existent system. The attack results in that particular port on the target being unable to respond to legitimate users. For example, to stop the users of a network receiving their incoming email an attacker could SYN flood port 110 (POP3) on the network's mail server.

To carry out this attack the synk4.c program was used. The attack program was used to SYN flood port 25 on the development server. Obviously the attack itself failed since this port is filtered by the firewall and there is no SMTP server running anyway. However, the incoming packets were still processed by the IDS.

The command syntax for synk4 is:

```
synk4 Source-IP Destination-IP Low-Port High-Port
```

If zero is used for the Source-IP field synk4 will use a different random unreachable source IP address for each packet it sends. The low-port and high-port fields allow the user to specify a range of ports to be flooded. To restrict the attack to just one port both fields can be given the same value. The following command was issued:

```
[root@titan]# ./synk4 0 server 25 25
```

This launched a stream of SYN packets to port 25 on the development server. Each packet had a different, spoofed, unreachable source IP address. Whilst running, synk4 displays a counter to indicate how many packets it has sent so far. Even on a dial-up connection the counter goes up much faster than is needed to successfully shut down a single port. The SYN flooder was allowed to send about 100 packets and then deactivated. As expected, the forged packets resulted in an alert from the NIDS. For the sake of brevity, only the parts of the email considered important are shown:

```
To: research@kaleton.com
From: ids@kaleton.com
Subject: Alert From NIDS

From Anomaly Detection Engine

For Incoming Packets To Destination Port 25, Average is
9.25472805368193, Standard Deviation is 2.2473957528348, and
Counted is 88

For Incoming Packets With The SYN Flag Set, Average is
5.35415029597069, Standard Deviation is 1.65944673996082, and
Counted is 94

For time taken for 100 packets,
Average is 200.37386327306138 seconds,
Standard Deviation is 17.95984673992142 seconds,
and Observed is 12.66944936596734 seconds

For the 100 packets this email refers to, the following IP
addresses were communicating with the server:

[There is a list here of about 100 (forged) IP addresses]
```

The email shows an unusually high number of packets addressed to port 25 and an unusually high number of packets with the SYN flag set. It also shows a huge decrease in the amount of time taken to observe 100 packets and hence a huge increase in volume of traffic. Whilst interpreting emails from the anomaly detection engine is not as easy as interpreting emails from the misuse detection engine it is still fairly obvious that a SYN flood attack against the SMTP port is underway. As the packets are spoofed there is no way for the NIDS to identify the true source of the attack.

5. Chapter Summary

It has been shown that during the course of a typical hacking attempt the prototype system will respond to several of the attacker's actions. The system administrator would be alerted as soon as information gathering has begun and can respond before the attacker moves on to the later stages of the intrusion attempt. It has been seen that the anomaly detection engine is good at detecting attacks and probes that involve the attacker sending many packets. For example, flooding DoS attacks and port scans both produce anomalous traffic. Attacks consisting of a small number of packets are less likely to be detected unless they are flagged by the misuse detection engine.

5. Evaluation and Further Work

In this chapter we look at the strengths and weaknesses of the prototype intrusion detection system that we have produced. Possible work to be done in the future is planned from the weaknesses found.

1. Strengths

This section describes the most noteworthy advantages of the features of the prototype system.

1. Strengths of the Misuse Detection Engine

We saw in chapter four that the attacks in the signature database are all detected by the misuse detection engine resulting in the attacker's IP address being emailed to the system administrator along with a simple, natural language, description of the attack. As misuse detection is performed on a packet by packet basis, these attacks are detected almost instantly. Real-time detection is important, as response time is a critical factor when dealing with an ongoing attack.

Adding new signatures to the database (such as when a new vulnerability has been discovered) is a simple task. For each new attack all that is needed is a line indicating whether the offending packet will be incoming or outgoing, which TCP flags will be set, any particular source or destination port used, and any string contained in the packet.

2. Strengths of the Anomaly Detection Engine

Anomalous traffic is assumed to be indicative of hostile behaviour by one or more external users. Events detected by the anomaly detection engine that would not be detected by misuse detection include: a sudden increase or decrease in the volume of traffic; fluctuations in the amount of traffic going to and from a given port; and fluctuations in the number of packets with a given flag set. Whenever a set of one hundred packets is found to be anomalous, precise details of the anomaly are emailed to the system administrator along with a list of all the external IP addresses appearing in that set of packets. The IP addresses are broken down by percentage of traffic to show how active each external user was during the period in question. This provides all the information necessary for a security officer to diagnose the cause of the anomalous traffic and make the necessary response.

3. Strengths of the Combination of Both Engines

The most obvious overall strength of the system produced in this project is the use of both misuse detection and anomaly detection. As was originally theorised in chapter three and then demonstrated in chapter four, the combination of these two methods allows detection of a wider range of attacks than either can achieve on its own. It was seen in chapter four that the anomaly detection engine is good at detecting activities that involve the attacker sending a large number of packets. This applies not only to existing attacks but to future attacks also. The misuse detection engine is excellent at detecting known attacks on a packet-by-packet basis. The two detection engines complement each other well and give the best chance of detecting any given attack.

4. Usage of System Resources

An important issue when deploying an intrusion detection system is how much of a drain it will have on system resources. This is especially true if, as in our case, the IDS is running on the same system that it is defending. For the prototype, the IDS itself is about 42Kb long and the statistical model, used by the anomaly detection engine to represent the network traffic, is approximately 1Mb. The model is kept both on disk and in memory. The attack signature database, also kept both on disk and in memory, is about 1Kb in size. This means a total of approximately 1Mb of disk space and RAM is used by the prototype system. As most systems today are equipped with 64Mb or 128Mb of RAM and at least 20Gb of hard disk the IDS is unlikely to cause memory or disk space problems. Apart from RAM and disk space, the other system resource that could be drained is processor time. On the development server, a 500 MHz Pentium, no noticeable decrease in speed resulted from running the IDS. This compares well with other systems. For example, SecureNet Pro from Intrusion Inc (http://www.extranet.co.nz/) which performs network-based misuse detection is advertised as having the following system requirements:

- Pentium II 400 MHz processor
- 128 MB RAM
- 250 MB disk space

5. Use of Generic Intrusion Detection Methods

Both the misuse detection engine and the anomaly detection engine could be easily adapted to take a different input. The techniques of pattern matching and statistical modelling could both be used on something other than TCP packets. Packets of other protocols would be easy to process but the system could be applied to data of other forms too. For example, pattern matching and statistical modelling could both be used to monitor command usage in a host-based IDS scenario.

6. User Interface

Originally, the intention was to have the NIDS log all attacks to a file in much the same way as Snort does. A user interface could then be written as a CGI script that could be accessed through a web browser. After providing authentication, the user could view a report of attacks in HTML form. However, this would require the user to login to the interface on a regular basis to check for recent malicious activity. It was decided that it would be far better to have the NIDS get in touch with the user than vice versa. This allows the user to make a much more timely response to intrusions.

2. Weaknesses

This section describes problems with the features of the prototype system and areas that require improvements to be made.

1. Weaknesses of the Misuse Detection Engine

The misuse detection engine operates on a packet-by-packet basis. Each packet captured by the sniffer is checked against each attack signature in the database. This is vulnerable to a way of attacking without detection. As some network devices can deal with larger packets than others, TCP allows packets to be fragmented if they are too large. This takes one packet and breaks it into two or more packets that are transmitted separately and then re-assembled on reception at the other end of the communication link. Those attack signatures that rely on a string in the data portion to detect attacks will not work if the packets are fragmented in such a way that the string is broken over two or more packets. For example, an attacker could use the MS IIS 5.0 .printer buffer overflow covered in chapter four without being detected by the misuse detection engine providing the string used in the signature is split over two packets.

Another, less serious, problem with the misuse detection engine is that after updating the attack signature database the IDS must be restarted. This in itself is not really a problem but it is likely to be forgotten by some users. If somebody relying on the system for security adds a new signature to the database when a new vulnerability is discovered but then forgets to restart the system he or she will have a false sense of security. The user will believe that the IDS will detect the new attack when in fact it will not until it is restarted. This problem arises due to the fact that the attack signature database is read from file when the IDS is started and stored in RAM to give optimum speed. Subsequent changes to the file will not be read into RAM until next time the system is started.

2. Weaknesses of the Anomaly Detection Engine

Whilst alerts from the misuse detection engine give precise details of which attack has been carried out and from which IP address, alerts from the anomaly detection engine are not quite as easy to interpret. Knowledge of the TCP protocol and specific attacks is required to work out the cause of the anomalous traffic. As stated at the beginning of the project, one of the main purposes behind automated intrusion detection is to remove the need for a human expert. The anomaly detection engine fulfils the other IDS requirement of removing the need for a human to check log files but it still requires a knowledgeable person to determine the likely cause of an anomaly.

There is also a way for an attacker to greatly reduce the value of alerts generated by the anomaly detection engine. If the attacker knows that his actions will cause an anomaly, he can add a lot of other packets at random so that the alert generated by the IDS will be almost impossible to interpret. If the junk packets all have spoofed source IP addresses it will not even be possible to work out which external user is causing the anomalies. For example, if an attacker carries out a port scan we would normally get an alert from the IDS showing a lot of packets from a single machine all addressed to different ports on our server. However, if the attacker mixed in a lot of packets with random flags set, addressed to random ports and with random spoofed source IP addresses it would be practically impossible to tell what is going on.

3. Dependence on Third Party Software

The intrusion detection system produced in this project requires the user to obtain and install the packet sniffer, TcpDump. Apart from the inconvenience, possible problems could result from differences in output between different versions and a possible lack of availability in the future. The prototype system was developed using version 3.6.2 of the software. It is possible that older or future versions may present captured packets in a slightly different way. For example some of the fields may be in a different order. This would cause the IDS to behave incorrectly.

3. Future Work

This section describes both possible solutions to some of the weaknesses from section 5.2 and also future research that could be done.

1. Adding Host-based Elements

The most promising area for future development of the system we have produced is adding host-based elements. This would result in a system combining both misuse detection and anomaly detection in both a network-based and host-based context! The aim would be to perform host-based intrusion detection on the activities of remote users who have established ftp or telnet sessions. This could again be achieved by examining the output of the packet sniffer. The commands being executed by remote users would be visible in the data parts of the packets. A statistical model of issued commands could be maintained by the anomaly detection engine in much the same way a model representing the network traffic is kept. Attack signatures consisting of particular commands known to indicate malicious behaviour (eg ‘cat /etc/passwd’) could be used by the misuse detection engine.

2. Handling Encrypted Traffic

The problem of performing intrusion detection on encrypted traffic could in part be dealt with by whichever server software the encrypted traffic is addressed to (eg the ssh daemon) co-operating with the IDS. If the ssh daemon shares the relevant encryption key with the IDS then it can decrypt and examine the data. If the IDS obtains the packet header and encrypted data from the sniffer and then obtains the encryption key from the relevant daemon it can

perform misuse detection as normal. This, however, would require modification of all the daemons that use encrypted traffic and so is probably not a practical solution.

3. Method of Capturing Packets

Instead of relying on TCPDump it would be worth writing some packet capturing software specifically for the IDS. This would eliminate the problems with relying on third party software already described. There is a library of packet capturing routines called libpcap available free of charge. However, this part of the IDS would have to be written in C if using libpcap and so would require recompilation on porting to another system.

It would also be useful to capture and process UDP and ICMP packets as well as the TCP packets already being dealt with.

4. Hardening the Intrusion Detection System

Further research should be done into making the IDS itself resistant to attack. If an attacker is aware that a network has an IDS running, he or she may decide to try to disable it before attacking the rest of the network. One area that should be looked at is preventing a user who has established a remote session to the server from shutting down the IDS. Currently, the IDS can be shut down by simply obtaining its process identifier and using the UNIX ‘kill’ command. Although this requires the user to be logged in as root it would still be a good idea to design some way of requiring an extra password to shut the IDS down.

5. Improvements to the Anomaly Detection Engine

The anomaly detection engine depends on creating and maintaining an accurate model of normal traffic in order to detect anomalous traffic. The task of detecting anomalous traffic could be performed with greater success if more properties of the network traffic were monitored and incorporated into the statistical model. For example, the anomaly detection engine could also monitor the size of packets, the number of fragmented packets and the spread of TCP sequence numbers used.

It would be useful for the anomaly detection engine to attempt to resolve IP addresses to host names prior to passing them to the console. This would make identifying the ISP of the attacker, and therefore who to report them to, quicker and easier.

6. Improvements to the Misuse Detection Engine

The misuse detection engine should reassemble fragmented packets before checking for a match with the signature database. TCP packets have sequence numbers and a flag indicating whether or not fragmentation has occurred. These could be used by the IDS to do the necessary reassembly. If this feature were added it would allow the misuse detection engine to catch attacks that have the signature string broken over two or more packets.

Extra syntax could also be added to the way attack signatures are specified so that attacks involving specific sequences of packets could be detected. For example, both the port scan and the SYN flood performed in chapter four are known attacks and so should be detected by the misuse detection engine. The current method of specifying signatures only allows for the detection of single packet attacks.

As with the anomaly detection engine, it would be useful for the misuse detection engine to attempt to resolve IP addresses to host names prior to passing them to the console.

7. Response to Intrusions

Some experimentation should be done with methods of dealing with attacks that could be performed by the IDS. For example, the misuse detection engine could automatically block the IP address of each detected attacker for a few hours by temporarily inserting a new firewall rule.

8. Improvements to the NIDS Console

Alternative methods of the NIDS sending alerts to the system administrator should be explored. The user could then choose one or more preferred methods. One possibility is for the NIDS to use SMS messaging to send alerts as text messages to the administrator's mobile phone. There are servers known as SMS gateways on the Internet which will receive data from another Internet host and forward it to a mobile phone network. These usually require a fee but would make a useful addition to the system. A potential problem is the large size of some of the alerts compared to the size of a mobile phone LCD display. Alerts from the misuse detection engine would probably be okay but alerts generated by the anomaly detection engine would often be too large to read on a mobile phone. A compromise would be to send a text message informing the administrator that an alert has been generated and that he should check his email. This would be useful to an administrator who is often away from his computer.

4. Chapter Summary

In this chapter we had a critical look at the prototype system that we have produced and talked about both its strengths and its weaknesses. We suggested some possible solutions to the weaknesses we found and also some areas for future research.

6. Summary and Conclusions

In this final chapter we summarise each of the previous chapters and present the conclusions we have drawn from the work carried out.

1. Summary of Previous Chapters

- Chapter 1: Introduction –The importance of intrusion detection was explained within the wider context of computer security.
- Chapter 2: Overview of Intrusion Detection – Past research into the field of intrusion detection was described and notable papers were reviewed.
- Chapter 3: System Specification and Design –A specification for combining both anomaly detection and misuse detection in a network-based IDS was proposed.
- Chapter 4: Attacks and System Response – An attempt to hack the development server was simulated and the NIDS' responses were observed.
- Chapter 5: Evaluation and Future Work – The successes and failures of this project were discussed and long-term research issues were identified.

2. Project Conclusions

The main conclusion drawn from this project is that it is both feasible and advantageous to include both a signature-based and an anomaly-based detection engine when producing a NIDS. Although the short amount of time available made it impossible to carry out very thorough testing of the prototype system, it was seen to detect many stages of a typical hacking attempt. Some actions were detected as misuse, others as anomalies. This suggests that further work on the prototype could produce a system capable of detecting a very wide range of attacks.

The implemented prototype system required minimal system resources. This showed that misuse detection and anomaly detection could be combined without resulting in a system requiring a lot of memory or a particularly fast CPU. The prototype did not even need a dedicated system. It ran on the same host that it was monitoring.

The prototype was also flexible enough to allow it to be applied to things other than TCP packets without major changes to the code. The system's modular design means that it could be easily extended to include more functionality.

Despite the challenge of dealing with encrypted traffic, it seems likely that network-based intrusion detection systems will continue to grow in popularity. As e-commerce continues to grow and web sites become more popular the field of intrusion detection is likely to concentrate on monitoring network traffic for external attackers rather than monitoring users who are logged in and executing commands. The popularity of web site defacement amongst a mostly teenaged cracker community is likely to remain for a long time. Compromised e-commerce sites can result in credit card information being taken and public embarrassment for the company. With attackers using tools that scan entire ISPs for a known vulnerability it is increasingly important that systems to detect this kind of activity are in place.

In the field of computer security, the constant race between attacker and defender is one that usually sees the defender struggling to keep up. Attackers discover a new vulnerability to exploit

or create a whole new attack paradigm (such as the shift to distributed attacks in 1999) and then weeks or months pass before the majority of system administrators take the necessary defensive steps. For this reason there would be great gains from incorporating the most sophisticated anomaly detection possible into future NIDS. An intrusion detection system that would reliably detect novel attacks would be a hugely powerful defensive weapon.

7. Appendix A – References

[1, White and Pooch] - Cooperating Security Managers: Distributed IDS, Gregory White and Vdo Pooch, 1996

[2, Cohen] - The Structure of Intrusion and Intrusion Detection, Fred Cohen, May 16th 2000

[3, Garvey and Lunt] - Model-Based Intrusion Detection, Thomas Garvey and Teresa Lunt, 1991

[4, Sebring et al.] - Expert Systems in Intrusion Detection: A Case Study, Sebring et al.

[5, Shaefer] - Employee Privacy and Intrusion Detection Systems: Monitoring On The Job, Shaefer

[6, Zenomorph] - Fingerprinting Port 80 Attacks: A Look Into Web Server, And Web Application Attack Signatures, Zenomorph

[7, Sasha] - A Strict Anomaly Detection Model for IDS, Sasha/Beetle, Phrack 56 0x0b

[8, Bejtlich] - Interpreting Network Traffic: A Network Intrusion Detector's Look at Suspicious Events, Richard Bejtlich, 14th May 2000

[9, Abren] - NIDS on Mass Parallel Processing Architecture, Wanderley J Abren Jr, Phrack 57 0x0c

[10, Rowland and Clark] - Automated Intrusion Detection : Theory and Practice, Leigh Rowland and John Clark, 1995

[11, Sasha] - Holistic Approaches to Attack Detection, Sasha, Phrack 57 0x0b

[12, Inella] - The Evolution of Intrusion Detection Systems, Paul Inella, November 16th 2001

[13, Anderson] - Computer Security Threat Monitoring and Surveillance, James Anderson, April 15th 1980

[14, Bace and Mell] - Intrusion Detection Systems, Rebecca Bace and Peter Mell, August 2001

[15, Deraison] - The Nessus Attack Scripting Language Reference Guide, Renaud Deraison

[16, NAI Inc] - CASL for Linux Red Hat 5.x Programming Guide Version 2.0, Network Associates Inc, 1999

[17, Axelsson] - Research in IDS: A Survey, Stefan Axelsson, August 19th 1999

[18, Ptacek and Newsham] - Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection, Thomas H. Ptacek and Timothy N. Newsham, January 1998

[19, Ryan et al] - Intrusion Detection with Neural Networks, Ryan, Lin and Miikkulainen, 1998

[20, Rain Forest Puppy] - A Look at Whisker's Anti-IDS Tactics, Rain Forest Puppy, December 1999

[21, arachNIDS] - Advanced Reference Archive of Current Heuristics for Network Intrusion Detection Systems, Max Butler, http://www.whitehats.com/ids/

[22, Psionic] - Psionic HostSentry, Information and Download, http://www.psionic.com/products/hostsentry.html

[23, Tripwire] - Tripwire home page, http://www.tripwire.com/downloads/

[24, Nazario] - Passive System Fingerprinting using Network Client Applications, Jose Nazario, November 27 2000

[25, Fyodor] - Remote OS Detection via TCP/IP Stack fingerprinting, Fyodor, October 18 1998

[26, Schwartz and Christiansen] - Learning PERL Second Edition, Randal Schwartz and Tom Christiansen, July 1997

[27, eEye] - Buffer overflow in Microsoft IIS 5.0, May 2001, http://www.eeye.com/html/Research/Advisories/AD20010501.html

8. Appendix B – Attack Signatures

The following five attack signatures are those that make up the database used by the misuse detection engine of the prototype system. They were chosen due to their frequency of use by the cracker community. A full version of the system would most likely use several hundred signatures. These five are, however, sufficient to demonstrate the principle.

```
EXTERNAL any INTERNAL 21 Ftp-probe S NULL
EXTERNAL any INTERNAL 23 Telnet-probe S NULL
EXTERNAL any INTERNAL 1080 Socks-Proxy-probe S NULL
EXTERNAL any INTERNAL 80 Frontpage-probe NULL 7674695f707674
EXTERNAL any INTERNAL 80 http-iis5-printer-eeye A 8bc483c01133c966b92001803003
```

9. Appendix C – NIDS Source Code

The perl source code for the prototype NIDS is available at http://www.kaleton.com/research/ids.pl.txt.