Detection of Network Threats using Honeypots

Peter Jackson

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

School of Computing

December 2005

Supervisor: Dr William Buchanan

Second Marker: Professor Peter Ross

Authorship Declaration

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

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

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

3. I have acknowledged all main sources of help.

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

5. I have read and understand the penalties associated with plagiarism.

Peter Jackson
01002890

Abstract

The increasing use of computer communication for many day to day tasks has resulted in a greater reliance on communication networks such as the Internet. The impact of a serious interruption to the operation of the Internet may have far reaching and costly consequences. The Internet has experienced several incidents caused by network worms, including an almost total shutdown by network as the result of the Morris worm in 1988.

This project covers the design, implementation and evaluation of a distributed honeypot system that provides the facilities to centrally log threat information. A system of this nature may collect information regarding a threat at the early stages of infection, allowing the possibility of an effective response to be deployed.

A number of software components have been developed in several programming languages including C, Perl and PHP. The prototype system run on a Linux based operating system.

Experiments were performed that demonstrated the systems ability to detect new threats within a short period of their first sighting.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Dr William Buchanan, Jamie Graves and Lionel Saliou for providing me with assistance, ideas and support. I am also grateful to Professor Peter Ross for giving me feedback at the Project Review and for being part of the marking team.

Thanks also must go to the authors and contributors to the software that I have utilised and extended. Special thanks should go to Niels Provos and Christian Kreibich, without their work much of this project would not have been possible.

1 Introduction

1.1 Project Overview

The overall aim of this project is to design, implement, test and evaluate a distributed honeypot system. Background research will be presented in order to understand existing techniques and themethods that may wish to be applied.

The dissertation presents a suggested set of requirements and design of a distributed honeypot framework. The framework will be developed into a design suitable for implementation. A prototype implementation based on the design will also be described.

It provides an evaluation of the effectiveness of the honeypot design and an implementation of a number of areas including, network traffic overhead, resilience against attacks, and the ability to detect previously unknown network threats.

1.2 Aims and Objectives

The aims and objectives of the project include:

- Review and evaluate existing research in the area of honeypots and current network threats.
- Design a distributed honeypot system that may be used to detect network threats.
- Implement a honeypot system and the associated testing framework to allow an evaluation to be undertaken.
- Undertake an evaluation of the design/implementation of the honeypot system.
- Make suggestions for future improvements and further research.

The level of success of the project will be based on these aims and objectives.

1.3 Project Context

Host based anti-virus software allow the detection and removal of suspected malicious software, such as network worms. Currently the primary means of detection used by anti-virus software typically involves the comparison against a database of known malicious code. A disadvantage of such an approach is that protection may only be provided against the threats included in the database. The process of incorporating a new threat involves capture and analysis that may have to be performed manually by human intervention. A delay of several hours, or more, may often be experienced between the initial sighting of a threat and it's inclusion in the database.

Signature based Network Intrusion Detection Systems (NIDS), such as Snort, may operate using similar techniques as host based anti-virus software. For this, a network communication channel is monitored for traffic that matches a set of signatures. An alert reporting a match with one of the signatures may be generated, and depending on the implementation, an attempt maybe made to block the threat.

A honeypot may be used to complement these defence techniques. Honeypots located within the network may be used to capture information about a threat so that defence measures may be configured to react to the new threat.

2 Theory

2.1 Introduction

This section provides background knowledge in areas of theory that will be developed in later sections.

2.2 Background Networking

Network systems may be conceptually represented by a multi-layered model. One such model is the OSI (Open Systems Interconnection) model defined by the ISO (International Standard Organization). Despite a failure of OSI model implementations to gain widespread use, the model may often be referred to when describing and attempting to gain a better understanding of network systems (Tanenbaum & Steen, 2002 p.58)

The emergence of the Internet brought with it an alternative layered model known as the Internet model (also referred to as TCP/IP). The Internet model is similar in many regards to the OSI model. A noticeable difference is the Internet model has an Application layer that takes the place of the OSI Application, Presentation and Session layers.

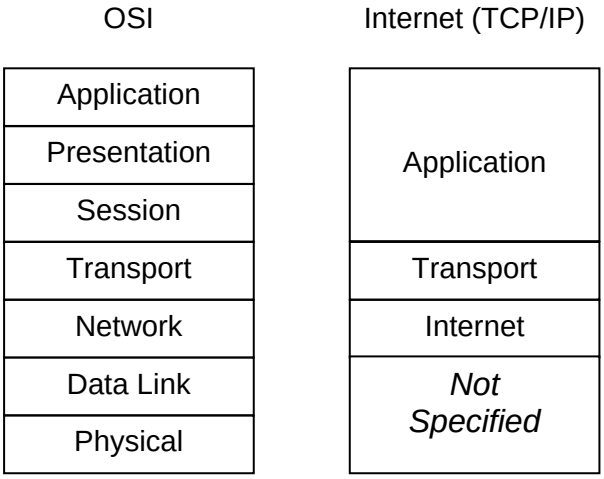

Figure 2-1 Comparison of OSI and Internet Stacks (based on Buchanan, 2005, p 13)

The function and characteristics of each of the OSI layers will not be covered in detail here. Tanenbaum and Steen (2002, p.61-68) provide an overview of the OSI model.

There are two general categories of transport layer protocol, connection oriented and connectionless. Connection oriented protocols such as TCP (Transmission Control Mechanism) provide a reliable connection between two end points, whilst connectionless protocols such as UDP (Universal Datagram Protocol) provide an unreliable connection. The choice between using connection oriented and connectionless transport may be made on an individual basis, however if connectionless is chosen error detection and correction will need to be handled by a higher level protocol.

In order to initiate a TCP connection a three-way handshake protocol (Figure 2-2) is followed. The TCP flags SYN and ACK are modified at each stage of the process. Following successful negotiation of a connection data may be sent between client and server. The client may disconnect the connection when the data transfer is complete by sending a packet with the FIN flag set.

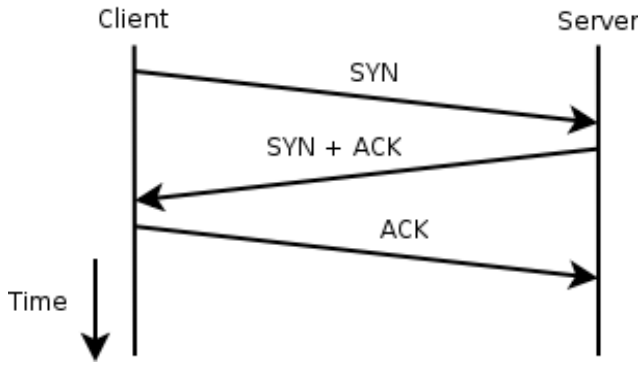

Figure 2-2 Three way TCP connection handshake (based on Tanenbaum & Steen, 2002, p 65)

2.2.1 Encryption

Public networks such as the Internet send data over shared communication channels. An opportunity exists for an eavesdropper with access to the communication channel or intermediate router to capture and view data. The data may be confidential or sensitive in nature, for example a credit card number or patient medical record.

In order to provide enhanced protection against viewing or modification whilst in transit encryption may be used to secure data. The layer at which the encryption is performed, is dependant on the implementation. A selection of encryption protocols in widespread use will be briefly described.

SSL/TLS
Secure Socket Layer (SSL), more recently known as Transport Layer Security (TLS) provides encryption between the Application and Transport layers. SSL may be used to encrypt an application level protocol that uses TCP as a transport protocol. For example, the protocol known as HTTPS is simply Hypertext Transfer Protocol (HTTP) over SSL. In order to use SSL encryption the client and server application programs must be SSL aware.

IPSec
IPSec implements encryption between the Transport and Internet layers. An advantage of IPSec is the ability to secure application protocols without any modifications to the application server and client programs. This has lead to IPSec being used to implement Virtual Private Networks (VPN).

2.2.2 Internet Addressing

The network protocol used on the Internet is known as Internet Protocol (IP). The most widespread version of IP currently in use is version 4, commonly written as IPv4. Each host is allocated a unique IP address that allows data packets to be sent to their destination.

IPv4 uses 32 bits to represent network addresses. The address is normally written as a set of 4 decimal numbers, in a format known as dotted decimal, for example 192.168.1.1

Earlier in the life of the Internet addresses were allocated in blocks of three sizes:

	Network Component Size	*Host Component Size*	*Number of Networks*	*Usable Host Addresses*
Class A	8 bits	24 bits	128	16,777,214
Class B	16 bits	16 bits	16,384	65,534
Class C	24 bits	8 bits	2,097,152	254

Table 2-1- Classes of Internet network addresses.

There are also Class D (multicast) and Class E (reserved) addresses but they are not relevant to our discussion so will not be covered further.

The allocation policy based on network classes has largely been discontinued for new allocations, as it does not allow for fine-grained allocation of network address. The replacement known as Classless Interdomain Routing (CIDR) allows networks of varying sizes to be allocated to an organisation

A proportion of the Internet address space is allocated to organisations for their use, however it is not actively used. For instance an organisation such as Napier University may have been allocated a class B network, with a possible maximum of 65,534 host addresses. It is likely that only a few thousand of the network addresses will be used. The unused addresses may be referred to by a variety of terms including "dark address space" (Krishnamurthy, 2004) or "network telescopes, blackholes and darknets" (Cooke, Bailey, Jahanian, Mao, McPherson and Watson, 2004)

The areas of dark address space present on the Internet may vary over time. For instance, an organisation may increase the number of hosts they wish to connect to the network resulting in hosts appearing on an area of previously dark addresses. The opposite may also be true in that a network previously populated with hosts may revert to being unused.

IPv6 is an updated version of the Internet Protocol that uses 128 bits for network addressing. Consequently, the total number of unique addresses is many times greater. The scale of IPv6 addresses is put into context by Zou, Towsley, Gong & Cai, (2005), assuming there are 1000 billion people on the earth, on average each person can own 2.3 million networks each with 4 billion networks the size of the current IPv4 Internet.

2.3 Firewalls

Public networks such as the Internet may allow hosts on separate networks to connect and attempt to gain access to services provided. In some cases, this universal connectivity may be undesirable. For example, a network security policy may restrict access to an Intranet site to only hosts within the local network, denying access to all other hosts.

A network device known as a firewall, located on the network border, may perform the implementation of the security policy. Firewalls may generally use a set of rules that determine if a packet should be permitted to pass or should be denied. Simple firewall rules may examine the Internet layer headers, for instance the source address of the packet. Rules that are more complex may examine transport or application level protocols, for instance allowing packets destined for TCP port 80 to pass through.

2.4 Denial of Service Attacks

The aim of a Denial of Service (DoS) attack is to prevent legitimate access to a resource, for example a web site. The exact technique used to perform a DoS attack may vary, but might involve consuming the networking bandwidth or host resources with deliberately generated requests. Legitimate requests may be severely delayed or lost altogether, resulting in loss of availability of the service.

An arguably more effective variation of a DoS attack is a Distributed Denial of Service (DDoS). Rather than a single host sending attack traffic to the target, multiple hosts distributed across the network may co-operate in the attack in order to create more attack traffic than a single host. The distributed nature of the attack may make blocking based on the source of the traffic ineffective, as the sources may be spread throughout the network.

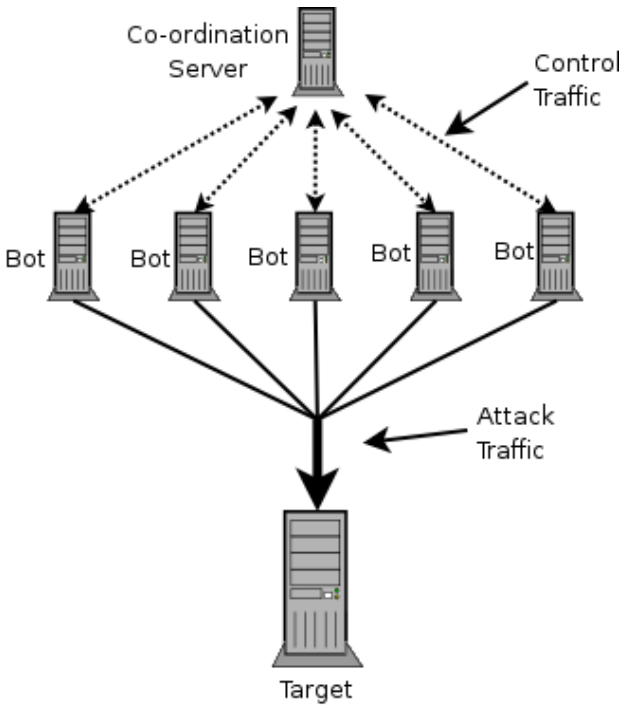

Figure 2-3 A Distributed Dentinal of Service

2.5 Honeypots

A Honeypot is a term given to a resource that has the primary purpose of attempting to gather information about security threats. The exact nature of the resource will vary but may take the form of one or more hosts connected to a network or the monitoring of an area of unused address space.

Within a network there may be production and non-production assets. Production assets provide services that are generally used by customers or employees. Failure of a production system is likely to have a noticeable negative effect on the operation of services, for example the failure of an e-mail server used by all employees. Non-production systems may generally not be used to provide services to the general user population. This enables tasks such as testing of system administration procedures or software changes without the risk of disrupting important services.

A Honeypot may generally be considered a non-production system. The presence of the Honeypot may also not be advertised to the rest of the network. An effect of these two assumptions is that in theory legitimate activity should not be directed toward the Honeypot, as there is no reason for connections to be made to a resource that isn't providing a production service.

By logging network traffic and system activity, information about probes and attempts to compromise the system can be gathered.

2.6 Network Worm Classification

- **Email (and other client application) worms**
  Propagation is by mean of email and similar application protocols such as Internet Relay Chat (IRC). They usually require user intervention in the form of running executable code.

- **Window File Sharing worms**
  Worms that spread using the Microsoft peer-to-peer networking service that is intended for file and printer sharing within a group of computers on a trusted network.

- **Traditional worms**
  These worms target specific vulnerabilities in operating systems and/or applications, allowing for propagation without use intervention. The first significant worm of this type was the Morris/Internet worm of 1988 (Spofford, 1989)The Morris worm exploited several vulnerabilities for which patches were not available. In more recent times worms have been released after a patch that prevented exploitation of the vulnerability. Kienzle. & Elder (2003) note that in general the time between the release of the patch and the exploitation of the vulnerability by a worm appears to be reducing.

There have been instances of a worm using multiple propagation techniques. For example the Nimda worm (Mackie, Roculan, Russel and Velzen, 2001) used four separate propagation methods spanning all of the above categories.

3 Literature Review

3.1 Introduction

Analysing data captured from a honeypot has a number of advantages:

- As there should be no legitimate activity, there are fewer privacy concerns regarding legitimate users' confidential data, as the investigator should not come into contact with sensitive information. An exception would be if data obtained from another compromise were to be stored on the honeypot.
- Greater monitoring can be performed without risking the performance of a production system.
- There is less pressure to bring a honeypot back into service following a compromise, allowing more time to preserve evidence for later analysis.

There are several types of honeypot that may be categorised based on the level of interaction that may be provided. High interaction honeypots are similar to real machines in that once an intruder has compromised the host security they are able to perform actions such as opening connections to other hosts. Low interaction honeypots do not allow this level of interaction, in fact they may be designed to prevent an actual compromise.

3.2 High Interaction Honeypots

High interaction honeypots generally consist of a standard installation of an operating system and associated service software. The level of security patches applied before deployment are determined by the honeypot operator. An unpatched system will be vulnerable to a wider range of threats as knowledge of exploitable vulnerabilities is greater. Deploying a system with all the published security patches is less likely to result in a successful compromise, although there is a small chance that successful exploitation of a previously publicly unpublished vulnerability may be observed (Spitzner, 2003). The honeypot may also be deliberately configured in a way that makes compromise easy, for example the super-user account may be protected with an easy to guess password such as "admin".

Modifications can be made to the operating system to allow closer monitoring of intruder activity. Information such as commands executed and any software tools that are downloaded by the intruder can prove useful in later analysis. Traditionally logging of this information on UNIX systems has been done by patching several key components of the operating system, such as the command line interpreter, know as a shell (Dornseif et al., 2004). The information collected by these modified binaries, including keystrokes are logged to a file or sent to another host on the network. In response to these techniques attackers may install their own binaries immediately following the comprise of the system, circumventing logging of this nature.

Network traffic that is not encrypted may be recorded by a network capture tool for later analysis. The increased use of encrypted equivalents in place of unencrypted services such as telnet and FTP means that analysis cannot be performed, as the key required for decryption is generally not available to the investigator.

In order to overcome the challenges imposed by increased use of encryption, specialised data capture methods have been developed. These tools work at the kernel level of the operating system, allowing access to the unencrypted data. One such tool is the kernel based data capture tool Sebek (The Honeynet Project, 2003). Originally developed for the Linux kernel, Sebek has been ported to other operating systems including Microsoft Windows, Solaris, and OpenBSD. Sebek intercepts a system call, capturing the content of most data transferred to and from the host. The logged data is transferred via UDP packets to a logging server located elsewhere on the network. It is interesting to note that the techniques used by tools such as Sebek have in part been inspired by tools know as rootkits that an intruder may install on a compromised machine in order to collect user passwords etc.

Dornseif et al. (2004) present several weaknesses in Sebek, allowing the presence to be detected. Following a successful compromise and escalation to super-user privilege, an attacker can scan the system memory for evidence of the data structures indicative of Sebek. Another method involves causing an overload of the logging mechanism. A one byte `read()` results in a UDP packet of approximately 100 bytes. A non-privileged user can carry out several hundred thousand one byte reads per second. Comparison of the ping response time under normal and high `read()` conditions can suggest the presence of a Sebek. Practical tests have shown that the average round-trip to another host on the network can rise from 0.7 milliseconds under normal conditions to 4800 milliseconds when a Sebek host is subjected to a test involving large amounts of `read()` calls (Dornseif et al., 2004)

For convenience reasons high-interaction honeypots may be run within a virtualisation environment. It may be trivial to create a new instance of an existing virtual machine image, allowing a compromised honeypot to be taken down for analysis and immediately replaced with a fresh non-compromised honeypot. Holz & Raynal (2005)present several methods to detect if the VMware virtualisation software is being used. The simplest of these involve examining the network card address to determine if it is in the range of address allocated to Vmware. Although the use of VMware does not necessarily mean that the host is a honeypot it may raise an attacker's suspicions.

It may be concluded from the available evidence that attempting to conceal the tools required to implement a host based high interaction honeypot may be problematic.

3.3 Low Interaction Honeypots

As with higher interaction honeypots the aim is to provide hosts and services on the network for probing and compromise attempts. In contrast, emulation is usually used in place of real operating system and server software. Using a set of predefined responses a service such as Simple Mail Transport Protocol (SMTP) server can be simulated. Although the simulated server behaviour closely matches a real server, they may often not carry out the expected action, for example the SMTP server may not actually send email.

Inevitably there will be differences between the simulated service and the real thing. For example authentication information may not be verified, allowing access with any

authentication credentials. This may raise the suspicions of a competent human attacker, leading them to believe something is amiss. Less intelligent threats such as self-propagating worms may not be able to make the distinction.

Provos (2004) describes an implementation of a low interaction honeypot that may allow a large number of virtual hosts to be simulated by a single physical machine. Each emulated host can simulate the network stack of an operating system. This allows the rapid configuration of multiple hosts with distinct "personalities")

3.4 Honeynets

The term honeynet describes an architecture that developed from the original honeypot concept. A honeynet as defined by Levine, LaBella, Owen, Contis and Culver (2003) is "a network, placed behind a reverse firewall that captures all inbound and outbound traffic".

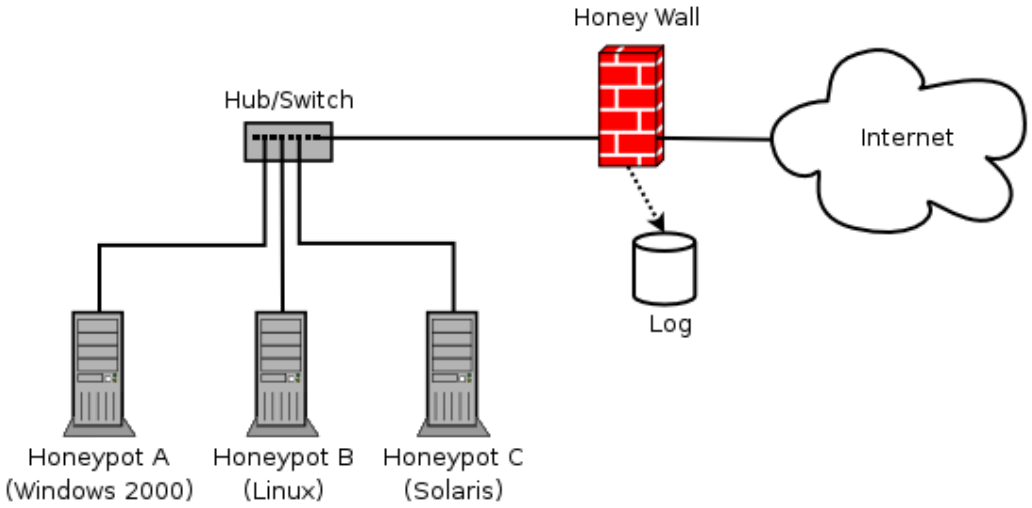

Figure 3-1 Typical Honeynet architecture

A high interaction honeypot poses a risk of being used by an attacker to inflict damage on third parties. To reduce the potential for such damage a honeynet incorporates several safeguards. As shown in Figure 3-1, all network traffic must pass through the honey wall. The honey wall is a firewall that protects the external network from attacks launched from honeypots within the honeynet. Incoming traffic to the honeynet is usually unrestricted in order to allow the honeypots full exposure to outside threats, hence the term "reverse-firwall". Denying all outgoing connections initiated by a host within the honeynet to the outside network would provide protection against attacks being launched from the honeypot. A downside may be that an intruder may easily detect such restrictions, as any connections the attacker attempts to make outside the honeynet will be blocked.

As described by Spitzner (2003) imposing a limit of outgoing connections (typically in the region of five to ten connections per hour) may be able to prevent most Denial of Service attacks and scanning. An attacker can determine if a connection rate limit has been applied by attempting to make a number of connections to outside hosts. If failures

are experienced after a number of connections then it is likely such a policy is in force (Dornseif, Holz & Klein, 2004).

Rate limiting may not be able to prevent the full spectrum of attacks. Attacks that target specific vulnerabilities only require a small number of connections to crash or compromise remote machines. The honeynet project has incorporated a modified version of the Snort Intrusion Detection System (IDS) known as Snort-Inline into the honeywall. Instead of simply generating alerts as most traditional IDS systems, it may also attempt to neutralise the threat. Although dropping the packet would prevent the attack, it may be obvious to an attacker. Snort-Inline makes a modification to the outgoing packet. For example many attacks again UNIX based systems contain the text `/bin/sh` in order to allow the attacker to execute commands. If this is replaced with the text `/ben/sh` then the attack will most likely fail.

The weakness of such an approach is that new threats not yet included in the IDS signatures or those obfuscated in order to avoid detection can slip through undetected (Spitzner, 2003). As a consequence a honeynet incorporating high interaction honeypots requires constant monitoring by a skilled operator for signs of abuse, resulting in a relatively high resource requirement.

3.5 Network Based Honeypots

Another related technique involves the monitoring of dark address space. These areas of address space should not have legitimate traffic entering them, therefore traffic may be the result of backscatter from spoofed source address, scanning by worms or other network probing tools. (Cooke et al., 2004)

Several approaches can be taken with regards to dark address space monitoring. (Cooke et al., 2004) describe an infrastructure known as the Internet Motion Sensor (IMS) that collects data from sensors listening to areas of dark address space. In addition to passive logging of traffic IMS responds to TCP requests. Responding to the initial connection attempt allows greater information to be collected, allowing the payload of TCP worms such as Blaster (Dougherty, Havrilla, Hernan & Lindner, 2003) to be captured.

Krishnamurthy (2004) presents a method for tracking suspicious traffic with the use of the multiple mobile network honeypots. A number of co-operating autonomous systems (AS) share information about the location of honeypots, in the form of a block dark address space. Each honeypot is advertised to the other co-operating AS for a duration determined by the address block administrator. During the advertisement period any traffic destined for the address space of one of the honeypots can be considered suspicious. By monitoring traffic passing through their links a co-operating AS may be able to identify the traffic closer to the actual source. This may be helpful when the attacker is using forged source network addressees in an attempt to hide the true source of the attack.

Honeypots can be applied to the detection of malicious network traffic and automated generation of Intrusion Detection System (IDS) signatures. Honeycomb (Crowcroft & Kreibich, 2004) applies pattern detection algorithms to packet header and payload of

network traffic captured by a honeypot to generate IDS signatures for threats. The analysis is simplified by monitoring a honeypot, as the differentiation between legitimate and malicious traffic has already been carried out.

Honeypots also have applications in education. A honeypot, with appropriate safeguards, can be used to provide students with an opportunity to learn about current threats. Jones and Romney (2004) observed a noticeable increase in student interest in security after involvement in a honeypot project, further work would need to be carried out to confirm if this is in fact the case.

3.6 Legal Issues

There are a number of legal issues surrounding the use of honeypots. There may be at least three legal areas that need to be considered in regard to honeypots under US law (Salgado, 2003). These include:

- Monitoring communication without appropriate permission of the communicating parties
- Liability for damage caused to third parties by attackers operating from a honeypot.
- Entrapment

In addition there is the possibility of intruders using a honeypot to store information such as stolen credit card numbers or to distribute illegally copied software (Salgado, 2003).The operators of a honeypot capable of data storage need to closely monitor for signs of such activity and intervene when such an event happens.

The applicable laws vary between legal jurisdictions, something that may be considered acceptable in one country may not however be legal in another country. The situation may be further complicated as multiple legal jurisdictions could potentially be involved. For example the suspected intruder may be located in a different jurisdiction to the honeypot.

3.7 Distributed Alert Data Systems

There is evidence of several existing systems that incorporate a centralised logging server. Two examples of such systems are DeepSight Analzyer (Symantec Corporation, 2004) and DShield.org (DShield.org, 2005).These two systems provide members of the Internet community with a client program that may be used to submit reports of suspected intrusions. The source of these reports may be typically firewall or IDS logs. As much of the submitted data may come from production systems and networks the theoretical risk of legitimate traffic being reported as suspicious may be higher than data obtained from a honeypot.

4 Design

4.1 Introduction

In general honeypot systems may be considered reasonably self contained. Monitoring and analysis may be carried out locally with results being fed back to the wider security community, for example in the form of white papers published by the Honeynet Project (The Honeynet Project, 2005) and others.

A perceived barrier preventing more widespread use of distributed honeypots is the differences of policy between the organisations that might become involved in a data sharing (Dagon et al., 2004). In designing such a system, consideration should be given to allowing organisations that do not have complete trust in each other to participate.

Thought also needs to be given to the participation of a hostile organisation that attempts to disrupt the operation or results generated by the system (Krishnamurthy, 2004)

This section describes the design of a distributed honeypot network that aims to allow the collection of threat data from diverse areas of the network in a form suitable for later analysis. Analysis of the combined data may be carried out by a variety of filtering techniques. Following the completion of automated analysis a skilled human may carry out further analysis. The honeypot network will be refered to by the term Honeytalk, expressing the concept of honeypots sharing information by "talking".

As far as possible the design will be constructed using a framework approach that may allow future work to build upon the foundations outlined here. Such an approach is important when considering a heterogeneous network such as the Internet as a design heavily linked to particular implementation may not be readily transferable between operating platforms.

4.2 Aims

Several perceived barriers may restrict the use of distributed honeypot networks. In order to attempt to design a system that reduces some of these barriers a greater understanding must first be gained.

Disclosure of Traffic Data
In theory, only potentially malicious traffic should enter the honeypot, however this may not be the case in practice. Legitimate traffic, possibly including sensitive data, may be recorded by the honeypot. There may be a variety of causes for such behaviour including use of area of network address space that is believed to be completely unused, but was in fact legitimately used sometime in the past. In order to reassure organisations it may be necessary to take steps to ensure that possible disclosure of this information is limited to what is permitted by legal and policy constraints.

Resource Requirements
Unless an organisation’s primary purpose is to provide information technology security services or undertake research, the participation in a distributed honeypot network is

likely to be considered a non-core activity. For example an Internet Service Provider (ISP) has the core business objective to provide network connectivity to their customers. Participation in a system that may allow threats to the security of the network to be detected may be seen as less of a priority when allocating funding, despite the potential to contribute to protection of the core business activity.

Because of limited funding, resource availability both in the form of equipment and staff time may be limited. This should be reflected in the design of the system to prevent barriers to participation.

In the context of this project, resource constraints were also imposed. Equipment had to be available within the existing environment or be obtained with little or no financial cost. Full implementation of the presented design is not a requirement but it is desirable to be able to implement a sub-set of the design using the available resources.

Acceptably Low Risk Level

As discussed previously honeypots may potentially allow an intruder to carry out attacks on third parties. It is possible that the organisation operating the honeypot will be held accountable for damage inflicted. As much as possible the design should aim to reduce the risk of the honeypot being abused.

4.3 Honeytalk Architecture Overview

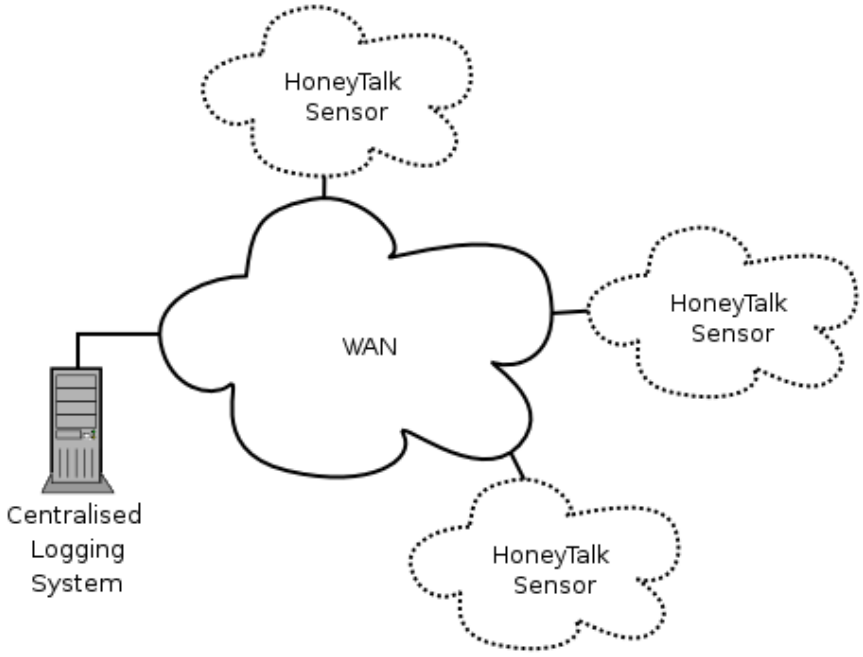

Figure 4-1 Architecture of Honeytalk system

The Honeytalk system may be considered as several sub-systems that will be described in the following sections.

- Sensors
- Data Recording Format
- Transport Mechanism
- Centralised Logging System

4.3.1 Sensors

Sensors, also known as honeypots, are located throughout the network in an area of previously unused address space. As discussed previously any traffic entering the sensor may be considered suspicious. The hope is that network threats that utilise scanning techniques will attempt to connect to the sensor.

There are several possible sensor architectures that can be applied to this situation. Each possibility will be briefly described and the suitability for use in Honeytalk discussed.

Passive Recorder
This is a very simple sensor that just records all incoming traffic. No response is generated in reply to incoming traffic, so from the attacking agent's viewpoint the network is "empty". The effective data gathering potential of such a sensor is limited to threats that use non-connection originated protocols such as UDP. Threats that use a connection originated protocol such as TCP would not be fully detected, as an active response is required in reply to the initial connection before the main payload is sent. (Cooke et al., 2004) The advantage of such an approach is that the overhead required to

passively monitor a network is relatively low, resulting in modest hardware requirements. The risk of comprise of the monitoring platform may be considered relatively low as there is no services, either real or emulated, exposed to the network. There have however been cases of vulnerabilities being found in tools that are used to implement such an approach (Manion, 2002).

Active Responder
By taking the concepts of a passive recorder and incorporating a simple responder mechanism it becomes possible to capture the payload of threats using connection originated protocols. As with passive recording, resource requirements may be considered relatively modest. The risk of compromise may be similar to that of the passive recorder, but the responder component poses an additional risk.

Low Interaction Honeypot
A low interaction honeypot may provide a simulated network stack and set of services. Simulation of a host network stack may be considered less resource intensive than a high interaction honeypot. Provos (2004) suggests that two class C networks (up to 508 hosts) may be simulated using honeyd with a single physical 666Mhz Pentium III host. Achieving a similar number of high interaction honeypots hosts may have much larger resource requirements.

The risk of comprise of low interaction honeypots may at least theoretically be relatively low. However an exploitable vulnerability in the network stack or service emulation components, may lead to compromise of the host running the honeypots.

High Interaction Honeypot
By using a number of high interaction honeypots a relatively high degree of realism may be provided. Even with the use of virtualisation technologies such as VMWare to allow multiple virtual hosts to exist on one physical host the hardware requirements may be relatively large. A compromise suggested by Dagon et al (2004, p 6) is to use the multiholming capabilities of operating systems to assign up to 32 IP addresses to a network interface. From a skilled attackers viewpoint a host with a large number of IP addresses assigned to a single interface may raise suspicions, a network worm may not be as discriminating.

High interaction honeypots can be designed in such a way that a full compromise may occur. This may pose a problem if used as a honeytalk sensor. Following compromise of the honeypot by malware the behaviour of that honeypot becomes unpredictable. For example, outgoing connection attempts may be made by a worm scanning for hosts to infect. The honeypot may no longer be trusted to provide consistent data and will need to be reset to a known state.

As discussed previously high interaction honeypots require careful setup and constant monitoring in order to ensure they are not used as a staging post for further attacks.

Decision
The low interaction honeypot approach has been chosen, as it allows application protocol simulation not provided by the simpler responder based methods. The higher possibility of misuse by an attacker and greater resource requirements rules out a high interaction honeypot in this situation. The choice was also influenced by the availability

of honeyd (Provos,. 2004) that allows the deployment of a network of low interaction honeypots on a single physical host . By using this existing software, effort can be concentrated on other elements of the system, such as the design of the transport mechanism.

4.3.2 Data recording format

The purpose of the data-recording format is to represent threat information in an efficient manor whilst attempting to meet the aims of the system

Transmission of unfiltered raw data
The simplest method of transmitting the collected sensor data is to send the full raw data to the central server. There is however an obvious drawback to such an approach, the amount of incoming data at the central server will be the sum of the traffic observed by all the sensors. Cooke et al. (2004) suggest that the raw data from a modestly sized sensor may be as much as one gigabyte per day. Whilst a well-connected host may be able to cope with data from a small number of sensors the bandwidth requirements may become excessive once the numbers and size of sensors start to rise.

During a period of higher than normal activity, for example during aggressive scanning by a worm, the volume of logging data transmitted may increase to a level that results in the saturation of communication links between the sensors and the central sever. In addition to the interruption of threat data to the central server, normal network traffic may be adversely affected. Rather than assisting in the reaction to the threat such an approach may actually amplify the network congestion caused by the worm.

Compression of the raw data before transmission could be used to reduce the volume of data. However compression requires a relatively large amount of processor time and may also increase the latency between data being received by the sensor and arriving at the central server.

Transmission of summary data
Reducing the volume of data that requires to be sent can be achieved by sending a summary of the traffic received by the sensor. One such method for summarising the data collected by a sensor is to apply a longest common substring (LCS) as implemented by Kreibich and Crowcroft (2004). By recognising a packet as being similar to another that has been previously observed by the sensor the need to transmit details of each packet may be reduced.

It is necessary to consider the data format used to represent the data recorded by the sensors. Typical details that may be relevant to later analysis include:

- Source Network Address - The network address of the host believed to be the source of the possible threat.

- Source Port - The UDP/TCP port that the possible threat originated from. It can be argued this information may not be particularly useful in the analysis process as the source host generally dynamically allocates the source port. However this is by no

means a certainty, for example, the Witty worm consisted of UDP packets with a constant source address 4000 and a random destination port. (Chien, 2004)

- Destination Network Address - The network address of the host that the threat was destined for. As this will usually be an address of one of the honeypots, care must be taken to limit the availability of this data to members of the honeytalk network, as it may lead to the publishing of the honeypot address space.

- Destination Port - The destination UDP/TCP port. This will typically be static for a particular threat. For instance a worm may target a specific vulnerability present in a server that listens on a well known port.

- Protocol - The Transport layer protocol that the possible threat uses. Typical value may include TCP, UDP or ICMP.

- Payload - The complete or partial payload of the packet. This may incorporate application level headers. Depending on the protocol involved this data may be binary data that cannot be represented by printable characters.

- Flags - Protocols such as TCP incorporate a set of flags that are used for a variety of tasks including the management of the connection state.

Representing threat data using XML has a number of possible advantages. Tools for transforming or extracting XML are readily available. Assuming that care is taken when specifying the format the representation should not be restricted to a specific technology.

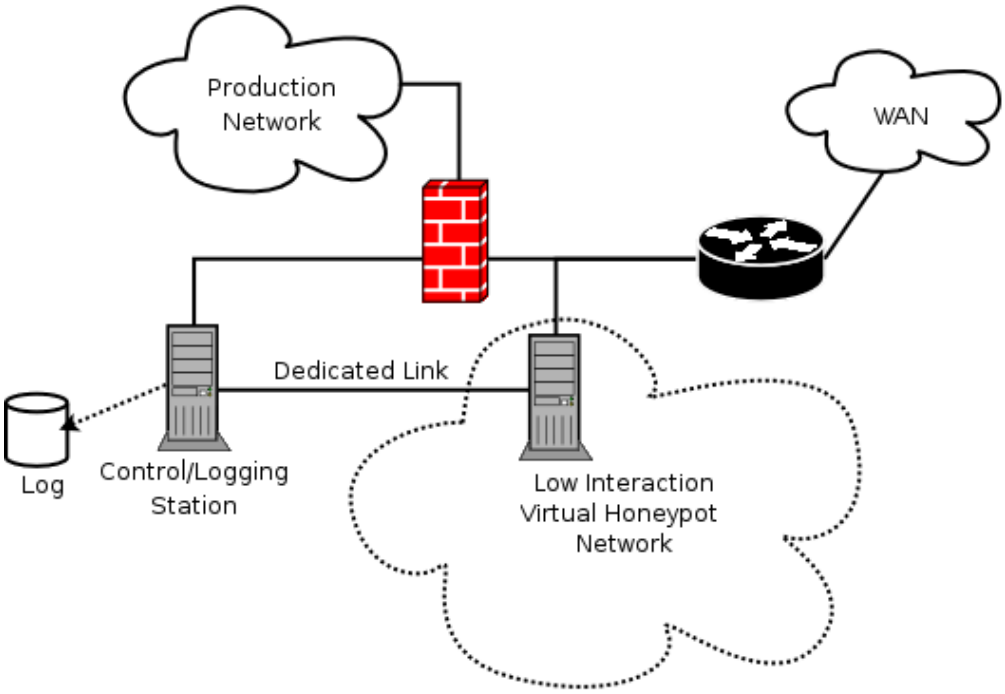

Figure 4-2 - Possible architecture of a HoneyTalk sensor

There is the possibility that a single host can provide the virtual honeypot network and the control/logging capabilities. Advantages of using two hosts is that a compromise of the virtual honeypot host would not directly result in access to the more potentially more sensitive data stored on the control/logging station.

4.3.3 Transport Mechanism

Requirements

Following capture and conversation into a data format suitable for transmission the threat data requires to be transmitted to the analysis server.

There are several requirements that need to be satisfied in order to achieve the aims of the system.

- Confidentiality - The threat data may be transferred over a public network such as the Internet. Measures need to be taken to ensure that the data is transmitted in a form that may not be easily read by a third party.

- Authentication - It should be possible to state with a high degree of certainty that threat data received by the analysis server originated from a member of the Honeytalk network. Furthermore it is important to be able to identify which user sent a particular alert, as it may later be found to be false data injected into the system to deliberately reduce the effectiveness of the system.

A possible method of improving the confidentiality of data is to perform encryption before transmission over the network. The data is then decrypted at the destination host.

There are several different form of encryption implementations that may be considered for use by the transport mechanism, these include:

IPSec
A possible drawback of may be the need to install software additional system network software as IPSec support is may not be provided by the base system. This may not be as much of a concern as in the past as several widely deployed operating systems and network devices incorporate IPSec support. For instance Microsoft Windows 2000, XP and Server 2003 incorporate support (Microsoft Corporation, 2004) . Although IPSec support is available for Linux based operating systems, past personal experience has shown that significant difficulties may be encountered.

SSL/TLS
SSL is generally better suited to providing encryption for a single application protocol rather than complete encryption at the network layer between client and server.

Username and password
The authentication element may be provided by a simple username and password combination allocated to each client of the system. Unfortunately experience has shown that using passwords as the primary means of authentication may be ineffective.

Summers and Bosworth (2004) discuss many of the problems that have been encountered as a result of relying on passwords for user authentication.

In the situation of Honeytalk the circumstances are a little different from a typical user authentication environment. The username and password would be entered in a configuration file once by the honeypot operator just before deployment of the honeypot. As a result the password could be of a greater complexity and length as there is not a requirement for the memorisation and regular entering of the password.

Another possible difficulty regarding the use of passwords is that of initial distribution of password from the administrators of the central server and the clients. Sending passwords in unencrypted email may be considered insecure as an eavesdropper may simply read the contents. Using an "out of channel" communication method such as postal mail or telephone for password distribution may make the job of an eavesdropper harder, as it would be necessary to compromise the security of one of these other communication networks.

The decision has been made to use SSL, combined with client certificates.

Certificate Infrastructure

The choice to use public key cryptography for both encryption and authentication requires consideration to be given to associated key management infrastructure.

When a new client wishes to join the network they need to first generate a private key and corresponding public key. A certificate request is signed with the client private key and transmitted to the Certifying Authority (CA). The signing request is not security sensitive so it may be transmitted using an insecure communication method. On receipt of the signing request the CA needs to verify that it originated from a legitimate source. Failure to properly verify the signature request may result in a potential attacker gaining access to the network.

Once the CA has carried out the necessary verification of the authenticity of the certificate request it is signed using the CA's private key and returned to the client. The client combines the signed certificate request with their private key to form a client certificate suitable for authenticating to the server.

An individual that has access to the CA private key may have the ability to sign arbitrary certificate requests, including certificates that appear to be from existing collaborating organisations. In order to reduce the risk of comprise of the private key a higher level of protection should be provided to the hardware and software used to sign certificates in addition to the actual key. Chokhani and Ford (1999) discuss various measures that may be used to secure the environment, including enhanced physical security and auditing. It should be remembered that in the case of Honeytalk the cost of a compromise may be relatively low, so the resources used to protect against such an event should reflect this.

4.3.4 Analysis System

Data collected by the sensors is periodically sent to the analysis system. The data is stored, allowing automated analysis to be carried out. Results of analysis may include a summary of network activity across the sensors with attention drawn to suspicious activity. The analysis system is also responsible for managing information about the sensors. For the purposes of this project the analysis will not be performed. The data will be stored in a format that would allow analysis to be performed.

5 Implementation

5.1 Introduction

It should be stressed that the system implementation outlined in the following section is a prototype to demonstrate how a distributed honeypot system may operate. It is not a complete implementation of the design presented previously.

In order to simplify the implementation of the prototype system, existing protocols and software have been used as much as possible.

Rather than continuing to describe the system as four separate sub systems: Sensors, Data Recording Format, Transport Mechanism and Analysis System, it will be considered primarily in terms of client and server portions. The reasoning for this change is that implementations of sub-systems such as the transport mechanism are closely integrated into other subsystems, making it easier to discuss them together.

5.2 Honeytalk Client

As discussed previously honeyd may be used to provide a number of low interaction honeypots that form a virtual network. The topology of the virtual network is defined in a text based configuration file that is read during initial start-up.

By default honeycomb generates output in the form of Snort rules that are periodically written to a text file. It is believed that shortcomings of this output implementation that make it unsuitable for direct use in the Honeytalk system, these include:

- The signature rules are appended to the end of the file after the rules created by the previous run of the output module. In order to extract only the most recent set of signatures, it would be necessary to determine the section of the file to be transmitted.

The snort signature output format is ideal for direct use by the Snort NIDS. However, in this situation the output may have further analysis performed on it at the central server.

The `honeycomb` code was written in such a way that allows alternative output mechanisms such as a database to be developed without the requirement for changes to the core software.

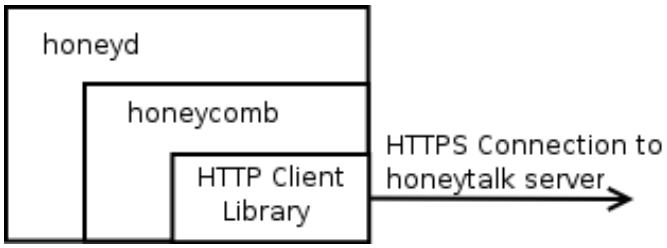

Figure 5-1 Block diagram showing transport mechanism embedded into honeycomb

Consideration was given to incorporating the transport mechanism directly into the honeycomb output module (Figure 5-1) This would allow the signature set to be transmitted to the central server immediately following the generation of revised signatures. Unfortunately there is a downside of such an approach, incoming packets are dropped whilst the output module is executing. Opening a network connection to a remote host is a time consuming task that may take several seconds. It is clear that dropping packets for such an extended period of time is unacceptable.

An alternative solution illustrated in Figure 5-2 was devised that utilised a database to store alerts. A honeycomb output module was built which inserts alerts into a local database. As discussed above, the time spent logging to the database may result in an interruption to the operation of honeyd. The period of the interruption may be smaller in this case as the connection to a local database may be considered much less time consuming than connecting to a remote host.

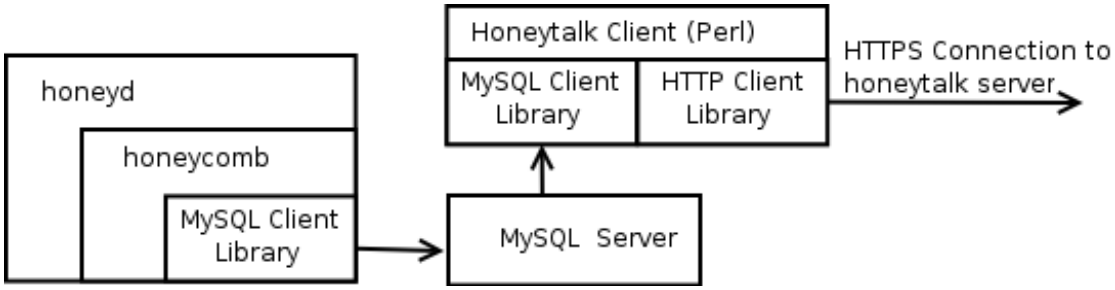

Figure 5-2 Block diagram incorporating an intermediate database.

Periodically the Honeytalk client script queries the local database and if necessary sends alert data to the server. A high level summary of the operation of the client script is provide in Figure 5-3

```
WHILE (1)
    Query the alerts database
    IF new alerts THEN
        Construct XML document containing alerts
        Open connection to server, using a client certificate for authentication.
        Send XML to server
        Close connection to server
    END IF
    Wait for a short period of time (default 10 seconds)
END WHILE
```

Figure 5-3 Pseudocode describing the operation of Honeytalk client.

ht_alerts

Column	*Data Type*	*Notes*
id	Integer	Unique identifier for the alert. Although unique within each client this is not globally unique as two clients may use the same

		identifier.
date	Time/Date	Time and Date of alert generation.

ht_alert_data

Column	*Data Type*	*Notes*
id	Integer	Automatically generated unique identifier
alert_id	Integer	Foreign key linking to ht_alerts.id
rule	Text	XML represention of the alert data

Table 5-1 Honeytalk client database table design

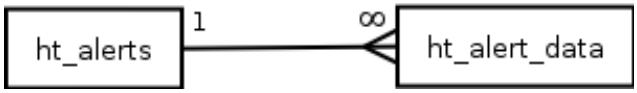

Figure 5-4 Entity Relationship diagram showing client database tables

5.3 Honeytalk Server

The role of the server is to store alert data received by clients in a form that can be analysed in real time or at a latter date. Implementation of analysis is out with the scope of this project so no effort has been made to implement such functionality within the server.

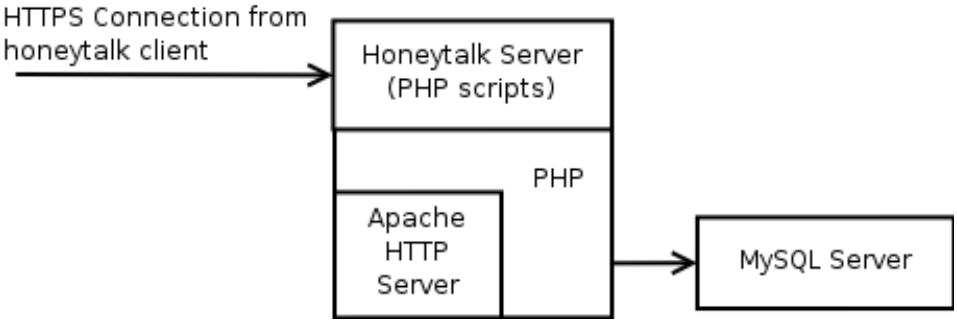

Figure 5-5 Block Diagram of Honeytalk Server

```
Process incoming connection from client
Gather client certificate information
IF client certificate is valid THEN
    Insert contents of client message into database
ELSE
    Send error message to client
END IF
```

Figure 5-6 Pseudocode describing the operation of Honeytalk server.

ht_srv_alerts

Column	*Data Type*	*Notes*
id	Integer	Automatically generated unique identifier
alert_date	Time/Date	Time and date the alert was received
client_id	Integer	
alert_data	Text	XML

Figure 5-7 Database table design for Honeytalk server.

5.4 Experiments

In order to evaluate the effectiveness of the system a number of experiments were performed.

Unless otherwise stated the hardware specification of physical hosts is as follows:

- Intel Pentium III 500Mhz Processor
- 128MB Main Memory
- 10Mbs Ethernet Network Interface Card
- IDE Hard Disk

The software configuration is based on the Knoppix live-cd (Knopper, 2005). Several additional software libraries required to support honeyd and honeycomb were incorporated into a customised Knoppix remaster. An image file of the knoppix CD used to perform the experiments is included on the CD accompanying this report.

5.4.1 Experiment 1

The aim of this experiment is to test the basic functionality of the system. Attack traffic is generated by the attacking host by using the simple netcat utility. In this particular case the attack was a message with the content "abcdefghijklmnopqrstuvwxyz", but equally may have been a network worm.

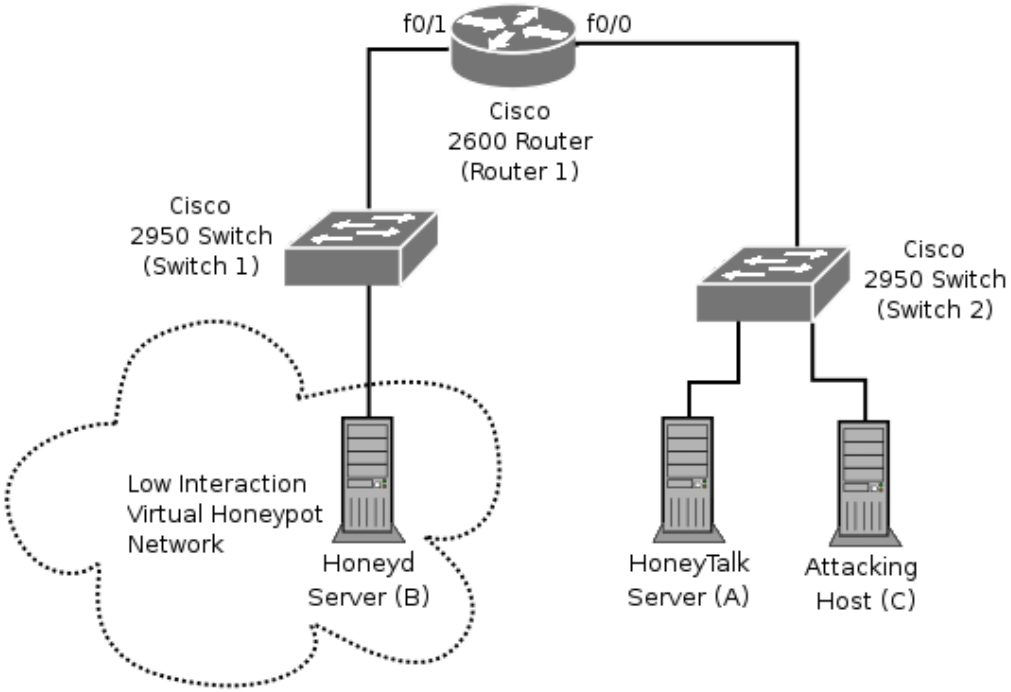

Figure 5-8 Experiment 1 Network diagram

	Network Configuration Details	Notes
Honeytalk Server (A)	IP Address 10.0.1.1 Subnet Mask 255.255.255.0 Default Gateway 10.0.1.254	
Honeyd Server (incorporating Honeytalk Client) (B)	IP Address 10.0.0.1 Subnet Mask 255.255.255.0 Default Gateway 10.0.0.254	Honeypots configured on hosts between 10.0.0.128 and 10.0.0.159
Attacking Host (C)	IP Address 10.0.1.2 Subnet Mask 255.255.255.0 Default Gateway 10.0.1.254	
Router 1	*Interface F0/0* IP Address 10.0.1.254 Subnet Mask 255.255.255.0	
	Interface F0/1 IP Address 10.0.0.254 Subnet Mask 255.255.255.0	

Table 5-2 Experiment 1 - Host configuration

5.4.2 Experiment 2

Attack traffic was generated

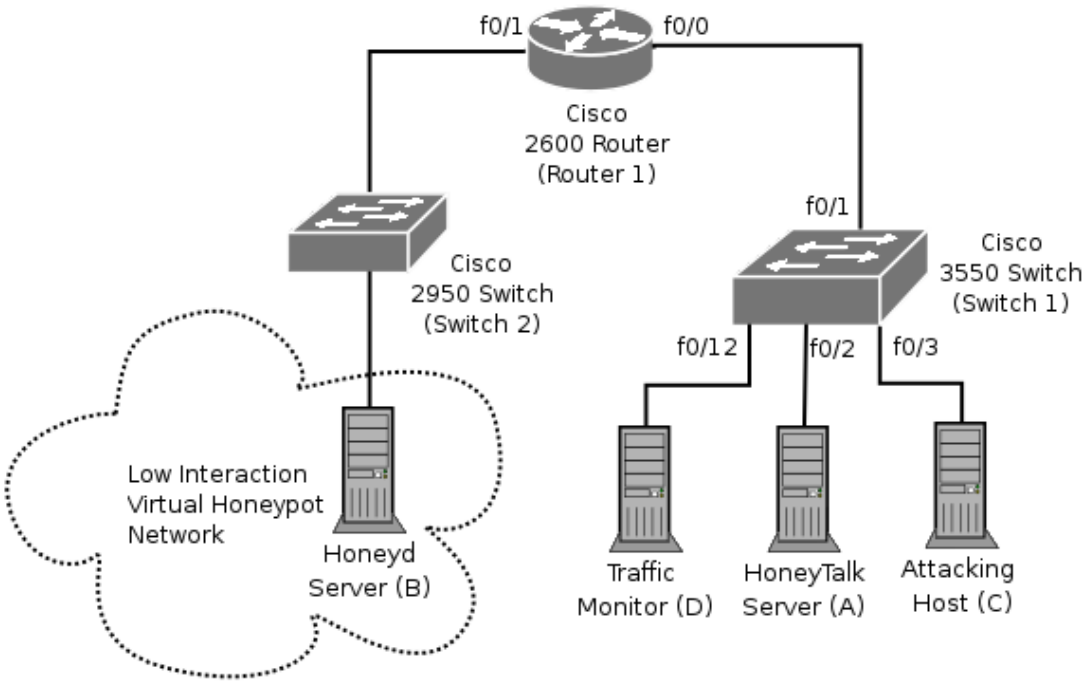

Figure 5-9 Experiment 2 Network diagram.

	Network Configuration Details	Notes
Honeytalk Server (A)	IP Address 10.0.1.1 Subnet Mask 255.255.255.0 Default Gateway 10.0.1.254	
Honeyd Server (B) (incorporating Honeytalk Client)	IP Address 10.0.0.1 Subnet Mask 255.255.255.0 Default Gateway 10.0.0.254	Honeypots configured on hosts between 10.0.0.128 and 10.0.0.159
Attacking Host (C)	IP Address 10.0.1.2 Subnet Mask 255.255.255.0 Default Gateway 10.0.1.254	
Traffic Monitor (D)	IP Address 10.0.1.3 Subnet Mask 255.255.255.0 Default Gateway 10.0.1.254	Connected to a switch port configured to act as a SPAN
Router 1	*Interface F0/0* IP Address 10.0.1.254 Subnet Mask 255.255.255.0	
	Interface F0/1 IP Address 10.0.0.254 Subnet Mask 255.255.255.0	

Table 5-3 Experiment 2 - Host configuration

The addition of a Traffic Monitor host allows the traffic between the honeytalk client and server to be monitored.

6 Results/Analysis

6.1 Introduction

This section presents experimental results and discusses improvements that might be considered for future experimental design.

6.2 Experiment 1

A short time after the attack packet was sent the alert shown in Figure 6-1 was received at the centralised logging location. The full attack packet payload is included in the content element. As a result this alert may be considered a good representation of the original threat.

```
<alert>
    <protocol>tcp</protocol>
    <source>
        <address>10.0.1.2/32</address>
        <ports>
            <port>any</port>
        </ports>
    </source>
    <destination>
        <address>10.0.0.129/32</address>
        <ports>
            <port>80</port>
        </ports>
    </destination>
    <date>Fri Nov 11 14:08:51 2005</date>
    <comment></comment>
    <flags>PA</flags>
    <flow>established</flow>
    <content>abcdefghijklmnopqrstuvwxyz</content>
</alert>
```

Figure 6-1 Alert generated by attack packet.

6.3 Experiment 2

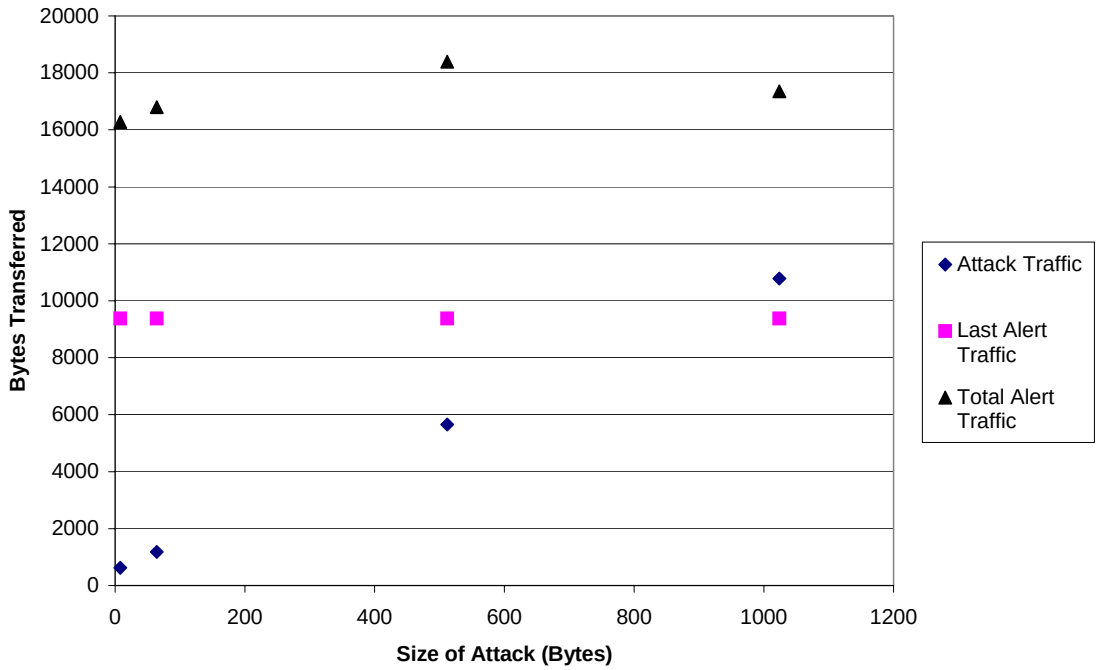

Figure 6-2 Data Transfer for various sizes of attack

Attack Size (Bytes)	Total Attack Traffic (Bytes)	Total Alert Traffic (Bytes)	Last Alert (Bytes)	Period from first attack to alert (seconds)
8	620	16,278	9,383	15.1644
64	1180	16,794	9,383	9.4320
512	5660	18,398	9,383	16.1587
1024	10,780	17,350	9,383	14.8147

Table 6-1 Experiment 2 results summary table

It should be noted that the total attack traffic is greater than ten times the attack size because headers are added at various network layers. For small attack sizes such as 8 bytes the majority of the total attack traffic volume consists of TCP, IP and Ethernet headers.

Honeycomb is configured to review and if necessary regenerate the signature pool every 10 seconds. Therefore, more than one set of alerts may be generated in the testing period. The exact occurrence of signature pool regeneration in relation to the attack traffic may vary.

It may be reasonable to expect that the sending of larger attack packets will result in an increase in alert traffic volume. The reason being that more storage may be required to represent the attack packet content in the alert. Table 6-1 shows alert traffic remained constant at 9,383 bytes despite the increasing attack packet size. Examination of the alert data sent to the server provides a possible explanation. Each run of the experiment resulted in nine identical alerts to that shown in figure Figure 6-3.

```
<alert>
    <protocol>tcp</protocol>
    <source>
        <address>10.0.1.2/32</address>
        <ports>
            <port>any</port>
        </ports>
    </source>
    <destination>
        <address>10.0.0.129/32</address>
        <ports>
            <port>80</port>
        </ports>
    </destination>
    <date>Mon Dec  5 20:08:51 2005</date>
    <comment></comment>
    <flags>FSRPAU210</flags>
    <flow>stateless</flow>
</alert>
```

Figure 6-3 XML representation of an alert generated by experiment 2.

The number of essentially identical alerts generated may raise doubts as to the effectiveness of honeycomb at producing summary information about a large number of similar packets. Before any conclusions may be drawn it is necessary to continue analysis of the experimental data.

In this case, absence of a content element suggests that the signature is based on the TCP/IP headers and not packet content. During the implementation of the XML output functionality several characteristics and headers including Time to Live (TTL) and Type of Service (TOS) were omitted. It was necessary to determine which, if any, of these omitted fields provided information that would distinguish the generated alerts. A subsection of the experiment was repeated using the original honeycomb file based snort output, in place of the XML/MySQL output modules.

Figure 6-4 shows a snippet of three alerts generated using the modified experimental procedure. As expected the alerts may be distinguished using one of the fields not included in the XML output, the ack field.

```
alert tcp 10.0.1.2/32 any -> 10.0.0.129/32 80
    (msg: "Honeycomb Mon Dec 5 20h49m29 2005 "; flags: FSRPAU210;
    ack: 455694444; flow: stateless;)
alert tcp 10.0.1.2/32 any -> 10.0.0.129/32 80
    (msg: "Honeycomb Mon Dec 5 20h49m30 2005 "; flags: FSRPAU210;
    ack: 1977130451; flow: stateless;)
alert tcp 10.0.1.2/32 any -> 10.0.0.129/32 80
    (msg: "Honeycomb Mon Dec 5 20h49m31 2005 "; flags: FSRPAU210;
    ack: 2118022621; flow: stateless;)
```

Figure 6-4 Alerts generated using the Snort output module

Another interesting feature of the alert shown in Figure 6-3 is that all valid TCP flags recognised by honeycomb appear to be set. Referring back to the data captured by the

traffic monitor does not show any packets with such a combination of flags set. A possible explanation for this seeming incorrect behaviour may be the fact that the attack packets did not have any TCP flags set. In normal conditions, a packet with the SYN (S) flag set is sent as the first stage in the connection negotiation process. It appears that due to a shortcoming in honeycomb, a packet sent with no flags set generates an alert with all flags set.

7 Evaluation

7.1 Attacks

There may be several theoretical attacks against the Honeypot system. The motivations for attacks may vary, but may include revenge from sections of the malware[1] development community who perceive the system as a threat to their operations. It should be noted that the term attack in this context does not necessarily mean that damage or disruption may result. It is used to describe any action that may result in reduced effectiveness of the distributed network or the analysis results.

7.1.1 Fingerprinting of the honeypots.

An adversary may attempt to locate honeypots by carrying out recognisance of the network. A method to reliably differentiate a honeypot host from a real host would need to be developed using techniques such as those outlined previously in the literature review . A Honeypot that does not generate a response to an incoming connection request may prove particularly difficult to differentiate from an area of completely unused addresses.

Performing a recognisance exercise on a large scale network such as the Internet may be considered sufficiently large that it is beyond the means of almost all individuals and groups of likeminded individuals. To systemically interrogate the address space of approximately 4 billion addresses at the optimistic rate of 1,000 per second may take approximately 50 days. There may be little chance of scanning on such a large scale going undetected either by the distributed honeypot network or the wider network administrator community.

If, despite the practical hurdles, a list of suspected honeypots were to be successfully collated there may be limits to the effectiveness of the use of the list in an attack. The dynamic nature of the Internet will mean that as soon as the list is complied it may require updating as real hosts and honeypots may appear and disappear over time.

An adversary may theoretically use a list of suspected honeypots to take steps to ensure that malicious traffic did not reach them. Implementation of a network worm that utilises such a black list may be impractical, due to the size and dynamic nature of the list. The storage requirements of all but the shortest of lists may outstrip the limited space available within the worm payload. Use of an alternative method to consult a blacklist may be considered by malware authors. Use of a secondary channel (Weaver et al., 2003) to query blacklist server(s) located on a host controlled by the attacker may be considered. Drawbacks of such an approach may include identification and shutdown of a black list server by authorities or overload the black list servers caused by a large number of connections made by worm instances.

7.1.2 Infiltration of the Honeypot Network

There may be the possibility of an attacker gaining access to the network in the role of a participating organisation. The method used to gain access may vary depending on the

[1] Malware is short for malicious software

policy of the honeypot network. In the case of a network with an open policy with no requirement to give details or evidence of identity the attacker may be able to request and be given a legitimate account to use the system. A network that requires participants to meet a set of requirements, for instance be a recognised UK university, and provide supporting evidence may require the attacker to take a different approach. For example the attacker may attempt to deceive a participating organisation into supplying details that will enable them to impersonate a legitimate organisation. A successful compromise of a Honeytalk client host may also enable an attacker to gain access.

Following successful infiltration a number of actions may be performed including sending of false alerts or excessive numbers of alerts. Sending a moderate number of false alerts may be done with the intention of reducing the quantity of results provided by the honeypot network, essentially reducing the overall effectiveness. A slightly less subtle approach may be to simply send an excessive number of alerts in the hope that the analysis system is unable to cope with the additional load. Counter measures such as denying access to the attacking organisation may prove effective against further abuse from that particular individual or set of individuals.

Reliably detecting when deliberate false alerts are being encountered may prove difficult. If only one sensor is reporting a specific alert then it could be considered suspicious, and more likely to be a deliberate false alert. Waiting for confirmation by a second sensor controlled by a separate organisation may help to provide greater resistance to attack of this nature. However, such an approach may reduce the effectiveness of the system at detecting a network worm at the early stages of propagation.

7.1.3 Denial of Service Attacks

The current architecture based on a single centralised server may be considered a considerable weakness. It provides a single relatively easy target for a Denial of Service attack (see page 12). During a sustained DoS attack connections to the central server may be delayed or fail altogether. Generally accepted defences that may be effective in reducing the impact of DoS attacks include a high capacity network connection combined with tuning of host network parameters.

7.2 Legal and Trust Hurdles

Many of the legal and trust hurdles discussed in section 4.2 are only partially addressed by the prototype. For example the captured data is encrypted in transit between the honeypot and centralised logging server, yet it may still include sensitive data.

The participation of a significant number of organisations distributed throughout a network such as the Internet may be unrealistic. It is proposed that such a system may have a greater chance of adoption in a network of similar organisations with similar network security polices. An example of such a network is JANET, a network linking United Kingdom academic and research institutions.

8 Conclusions

8.1 Achievement of aims/objectives

As discussed in Section 1.2 the project aims and objectives of the project are as follows:

- Review and evaluate existing research in the area of honeypots and current network threats.
- Design a distributed honeypot system that may be used to detect network threats.
- Implement a honeypot system and the associated testing framework to allow an evaluation to be undertaken.
- Undertake an evaluation of the design/implementation of the honeypot system.
- Make suggestions for future improvements and further research.

The literature review examined research in honeypots and related areas. The main types of honeypots were described and the disadvantages and advantages of each discussed. Research into the main types of network worms was also undertaken and presented.

The implementation was successful, as demonstrated by experiment 1, generating a signature matching an unknown threat. In order for the system to be deployed successfully on a publicly connected network a number of improvements, including those described in section 8.2, may need to be implemented.

The aims and objectives were achieved to varying extents of completeness. For instance, a basic evaluation of the system was performed that was adequate to test the basic operation. However, the effectiveness when faced with a more realistic threat was not tested. As discussed in the Future Work section, it may be necessary to carry out further experiments in order to perform a fuller evaluation.

8.2 Future Work

Further evaluation in the system is necessary to gain a better understanding of the effectiveness in detecting a more realistic threat. For instance, an experiment that attempts to simulate the effects of a worm scanning for hosts to infect. A potential experimental procedure involves multiple hosts located throughout a network that are listening for incoming connections. On receiving an incoming "magic" packet, a vulnerable host will commence scanning for other vulnerable hosts to "infect". Altering the number, location and size of honeypots that comprise the distributed honeypot network may allow the investigation of factors that effect the relative effectiveness at detecting the threat. The use of virtualised hosts may help to reduced hardware requirement for an experiment of this nature. Consideration may also be given to a purely simulated experiment based on a mathematical model of worm propagation.

Consideration may wish to be given to data-hiding techniques, such as described by Llamas (2004) to send traffic between the honeypots and the analysis system. In doing so, it may make discovery of the location of a honeypot more difficult. A drawback of such an approach is that traffic flows will still appear between the client and the server,

effectively resulting in no discernible advantage over the current approach. The volume of data required to successfully hide a message may be many times larger than the data itself, resulting in increased data transfer volume.

The reduction of the data transfer from the client to a centralised logging server also needs further investigation. Sending only the alert data generated since the previous alert transmission may go a long way in reducing the volume of alert traffic.

Future work may wish to examine how failures of components of the system, in particular the transport mechanism, may be better handled. The current implementation uses a "best-effort" scheme that generally simply outputs an error message when an error is encountered. Retransmission of data following a network timeout when attempting to connect to the server may wish to be considered. However it is important to take into account that such an approach may result in duplicate alert messages being received by the server. (Tanenbaum & Steen, 2002, p.379)

Counter measures against false alerts and excessive alerts may wish to be investigated. A simple set of heuristics, based on alert transmission rate, may be applied to alert data. For example when an organisation that normally transmits an average of 10 alerts per hour, suddenly begins to send two alerts per second. The cause of the dramatic increase in alerts may be the result of a legitimate increase in honeypot activity. It may also indicate a possible excessive alert attack. Drawing the attention of a skilled human operator, or a specialist software analysis agent, to the increase in alert rate may be the most appropriate response.

Investigation into the effects of more widespread IPv6 adoption may also merit consideration. The extremely large number of network addresses may have implications on the effectiveness of a honeypot-based system, such as Honeytalk. The proportion of the total IPv6 network address space that could be practically monitored by a network of honeypots may be smaller than that of IPv4. As discussed by Zou et al.(2005) the immense size of the address space IPv6 may provide an effective defence again worms such as Blaster (Dougherty et al., 2003) and Witty (Chien, 2004) that find victims by random scanning.

It may be anticipated that a single sever may become a significant bottleneck in terms of network bandwidth and/or processing resources. Investigation into the possibility of using a decentralised model with analysis being spread across multiple network agents may wish to be considered.

9 References

Buchanan, W. (2005) *Advanced Security and Mobile Networks Course Notes*, Napier University, Edinburgh

Chien, E.(2004, March 22) *Symantec Security Response - W32.Witty.Worm* Retrieved December 1, 2005 from http://securityresponse.symantec.com/avcenter/venc/data/w32.witty.worm.html

Chokhani, S., Ford, W. (1999) *RFC 2527 - Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework* Retrieved December 5, 2005 from http://www.ietf.org/rfc/rfc2527.txt

Cooke, E., Bailey, M., Jahanian, F., Mao, M., McPherson, D. & Watson, D (2004) Toward Understanding Distributed Blackhole Placement. *Proceedings of the 2004 ACM workshop on Rapid malcode* pp. 54-64

Dagon, D., Qin, X., Gu, G., Lee, W., Grizzard, J., Levin, J. & Owen, H. (2004) HoneyStat: Local Worm Detection Using Honeypots. *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, France, September 2004.

Dornseif, M., Holz, T., & Klein, C. (2004). NoSEBrEaK - Attacking Honeynets. *Proceedings of the 2004 IEEE Workshop on Information Assurance and Security*, USA, pp. 1-7

Dougherty, C., Havrilla, J., Hernan, S., & Lindner, M. (2003, August 14*) CERT Advisory CA-2003-20 W32/Blaster worm.* Retrieved May 16, 2005 from http://www.cert.org/advisories/CA-2003-20.html

DShield.org (2005) *DShield.org -Distributed Intrusion Detection System* Retrieved December 5, 2005 from http:// www.dshield.org/

Hinden, R. (1995) *IP Next Generation Overview* Retrieved December 1, 2005 from http://playground.sun.com/pub/ipng/html/INET-IPng-Paper.html

Holz, T., & Raynal, F. (2005). Detecting Honeypots and other suspicious environments. *Proceedings of the 2005 IEEE Workshop on Information Assurance and Security*, USA, pp. 1-8

Jones, K. & Romney G. (2004) Honeynets: An Education Resource for IT Security. *Proceedings of the 5th conference on Information technology education*, USA, pp 24-28

Kienzle, D. & Elder, M. (2003) Recent worms: a survey and trends *Proceedings of the 2003 ACM workshop on Rapid Malcode* , USA, pp 1-10

Knopper, K. (2005) *Knoppix Information* Retrieved December 10, 2005 from http://www.knopper.net/knoppix-info/index-en.html

Kreibich, C. & Crowcroft, J. (2004) Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. *ACM SIGCOMM Computer Communications Review*, 34(1), 51-56

Krishnamurthy, B. (2004). Mohonk: mobile honeypots to trace unwanted traffic early. *Proceedings of the ACM SIGCOMM workshop on Network troubleshooting: research, theory and operations practice meet malfunctioning reality*, USA, pp. 277-282

Levine, J., LaBella, R., Owen, H., Contis, D., & Culver, B.(2003). The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks. *Proceedings of the 2003 IEEE Workshop on Information Assurance*, USA, pp. 92-99

Llamas, D. (2004) *Covert Channel Analysis and Data Hiding in TCP/IP* Retrieved December 5, 2005 from http://www.dcs.napier.ac.uk/~bill/PROJECTS/2004/david_llamas.pdf

Mackie, A., Roculan, J., Russel, R., & Velzen, M.V. (2001) Nimda worm analysis. *Security Focus, Incident Analysis Report, Version 2* Retrieved October 10, 2005 from http://aris.securityfocus.com/alerts/nimda/010919-Analysis-Nimda.pdf

Manion, A. (2002) Vulnerability Note VU#797201 tcpdump vulnerable to buffer overflow via improper decoding of AFS RPC (Rx) packets. *US Computer Emergency Readiness Team Vulnerability Notes Database* Retrieved November 17, 2005 from http://www.kb.cert.org/vuls/id/797201

Microsoft Corporation (2004) *Internet Protocol Security* Retrieved December 5, 2005 from http://www.microsoft.com/windows2000/technologies/communications/ipsec/

Provos, N. (2004) A Virtual Honeypot Framework. *13th USENIX Security Symposium*, USA, pp.1–14

Symantec Corporation (2004) *DeepSight Analzyer Overview* Retrieved December 5, 2005 from http://analyzer.symantec.com/AnalyzerIntro.html
Salgado, R. (2003). The legal ramifications of operating a honeypot. *IEEE Security & Privacy*, 1(2), 16-17

Spofford, E (1989) The internet worm program: an analysis *ACM SIGCOMM Computer Communication Review* , 19(1), 17-57

Spitzner, L. (2003). The Honeynet Project: Trapping the Hackers. *IEEE Security & Privacy*, 1(2), 15-23

Summers, W., Bosworth, E. (2004) *Password policy: the good, the bad, and the ugly* Proceedings of the winter international synposium on Information and communication technologies, Cancun, Mexico, 1-6

Tanenbaum, A., Steen, M. (2002) *Distributed systems :Principles and Paradigms* New Jersey: Prentice Hall

The Honeynet Project (2003, November 17) *Know Your Enemy: Sebek.* Retrieved May 16, 2005 from http://www.honeynet.org/papers/sebek.pdf

The Honeynet Project (2005) *Whitepapers* Retrieved May 16, 2005 from http://honeynet.org/papers/

The SANS Institute (2001, April 18) *Lion Worm* Retrieved November 17, 2005 from http://www.sans.org/y2k/lion.htm

Weaver, N., Paxson, V., Staniford, S, Cunningham, R. (2003) A taxonomy of computer worms *Proceedings of the 2003 ACM workshop on Rapid Malcode,* Washington, DC, USA pp 11-18

Zou, C., Towsley, D., Gong, W., Cai, S. (2005) *Routing Worm: A Fast, Selective Attack Worm Based on IP Address Information* Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp 199-206

10 Bibliography

Briesemeister, L., Lincoln P. & Porras, P. (2003) Epidemic profiles and defense of scale-free networks *Proceedings of the 2003 ACM workshop on Rapid Malcode*, USA, pp. 67-75

Caswell, B., Hewlett, J. (2005) *Snort Users Manual* Retrieved December 10, 2005 from http://www.snort.org/docs/snort_manual.pdf

Cisco Systems, Inc.(2005, September 26) *IP Addressing and Subnetting for New Users* Retrieved December 1, 2005 from http://www.cisco.com/warp/public/701/3.html

Housley, R., Polk, W., Ford, W., & Solo, D. (1999) *RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile* Retrieved Nov 8, 2005 from http://www.ietf.org/rfc/rfc2459.txt

Internet Assigned Numbers Authority (2005, April 12) Retrieved December 1, 2005 from http://www.iana.org/ipaddress/ip-addresses.htm

Maj, A (2005) *Apache 2 with SSL/TLS: Step-by-Step, Part 1* Retrieved October 10, 2005 from http://www.securityfocus.com/infocus/1818

Maj, A (2005) *Apache 2 with SSL/TLS: Step-by-Step, Part 2* Retrieved October 10, 2005 from http://www.securityfocus.com/infocus/1820

Maj, A (2005) *Apache 2 with SSL/TLS: Step-by-Step, Part 3* Retrieved October 10, 2005 from http://www.securityfocus.com/infocus/1823

Raynal, F., Berthier, Y., Biondi, P., Kaminsky, K. (2004a) Honeypot Forensics Part I: Analyzing the Network, *IEEE Security and Privacy*, vol. 2 (4), 72-78

Raynal, F., Berthier, Y., Biondi, P., Kaminsky, K. (2004b) Honeypot Forensics, Part II: Analyzing the Compromised Host, *IEEE Security and Privacy*, 2(5), 77-80

Appendixes

Appendix A - Description of modified/new honeycomb source files

Modified/New Source Files

Source Code File	*Change Description*
`hc_xml_printer.c`	Heavily based on the existing `hc_snort_printer.c` modified to output a XML document instead of a snort signature. The full functionality of the snort printer is not implemented. For instance seq (sequence numbers) and dsize (data size) are not included.
`hc_xml_printer.h`	Heavily based on the existing `hc_snort_printer.h`
`hc_mysql_logger.c`	Based on the existing `hc_file_logger.c` Insert alerts into a MySQL database using the MySQL C API.
`hc_mysql_logger.h`	Heavily based on the existing `hc_file_logger.h`
`hc_config.c`	Added several new configuration directives for switching between xml/snort signature format and file/MySQL output. Also added MySQL database connection settings such as host, port, database, username and password.
`hc_config.h`	Modified to include new configuration setting introduced into `hc_config.c`
`honeycomb.c`	Modified to call the relevant signature/output initialisation and callback registration based on the configuration directives.

New configuration directives

Directive Name	*Valid Values*	*Default*	*Notes*
logging_module	"file" or "mysql"	"mysql"	
sig_output_module	"snort" or "xml"	"xml"	
mysql_hostname	Valid Hostname	"localhost"	The hostname to connect to a MySQL server on. Note IP cannot addresses cannot be used. result in a
mysql_username †	Valid MySQL username	""	
mysql_password †	Valid MySQL pasword	""	Password used to connect to the database.
mysql_database †	Valid MySQL database	"honeypot"	The database to log alert data to. Database must have an existing ht_alerts and ht_alert_data table.
mysql_port †	Interger	3306	Port that the MySQL server is listening on

† Only relevant if logging_module is set to "mysql"

Appendix B - Honeytalk Client and Server Source Code

Client script - sendalerts.pl

```
#!/usr/bin/perl -w
  use config;

  use LWP::UserAgent;
  use Mysql;
  use DBI;

sub do_http_request {

  my $XmlData = $_[0];

  if (HT_HTTPS_ENABLED != 0) {

    if (defined(HT_HTTPS_PKCS12_FILE)) {
      $ENV{HTTPS_PKCS12_FILE}     = HT_HTTPS_PKCS12_FILE;
    }

    if (defined(HT_HTTPS_PKCS12_PASSWORD)) {
      $ENV{HTTPS_PKCS12_PASSWORD} = HT_HTTPS_PKCS12_PASSWORD;
    }

    if (defined(HT_HTTPS_CA_FILE)) {
      $ENV{HTTPS_CA_FILE} = HT_HTTPS_CA_FILE;
    }

  }

  # Create a HTTP user agent object
  $ua = LWP::UserAgent->new;
  $ua->agent("HoneyTalk/0.1");

  # Create a request
  my $req = HTTP::Request->new(POST =>
HT_HTTP_BASE_URL.'xmlAlertLog.php');
  $req->content_type('text/xml');
  $req->content($XmlData);

  # Pass request to the user agent and get a response back
  my $res = $ua->request($req);

  # Check the outcome of the response
  if ($res->is_success) {
    print $res->content;
  }
  else {
    print $res->status_line, "\n";
  }
}

  # connect to the MySQL DB
  $dbh = DBI->connect("dbi:mysql:".HT_DB_DATABASE.";host=".
HT_DB_HOSTNAME. ";port=".HT_DB_PORT,
                    HT_DB_USERNAME,  # username
                    HT_DB_PASSWORD # password
                      )
```

```
  || die "Cannot connect to database";

  # read the alerts from the MySQL DB and package them into a HTTP
  # request

  my $maxAlertId = 0;

  my $queryAlerts = $dbh->prepare('SELECT id, date, notes FROM
ht_alerts WHERE id > ?');
  my $queryMaxAlertId = $dbh->prepare('SELECT max(id) AS max_alert_id
FROM ht_alerts');
  my $queryAlertData = $dbh->prepare('SELECT rule, id FROM
ht_alert_data WHERE alert_id = ?');

  # as we have just been started up we don't want to send all
  # the alerts currently in the DB, as they have probably already been
  # sent to the server

  $queryMaxAlertId->execute();
  $maxAlertIdRow = $queryMaxAlertId->fetchrow_hashref();

  $maxAlertId = $maxAlertIdRow->{'max_alert_id'};

  my $currentAlertId = $maxAlertId;
  while (1) {

    # query the alerts table
      $queryAlerts->execute($maxAlertId);

    my $alertXMLData = "";

      while ($alertRow = $queryAlerts->fetchrow_hashref()) {

      $currentAlertId = $alertRow->{'id'};

      $alertXMLData = "<alerts>\r\n";
      print "Reading AlertId: $currentAlertId\n";

      # retrieve the alert data items associated with the alert
      $queryAlertData->execute($currentAlertId);

      while ($alertDataRow = $queryAlertData->fetchrow_hashref()) {
        $alertXMLData .= $alertDataRow->{'rule'};
        }

      $alertXMLData .= "</alerts>\r\n";
      do_http_request($alertXMLData);
    }

    $maxAlertId = $currentAlertId;

    print "Tick...\n";
    sleep HT_ALERT_CHECK_PERIOD;
  }
```

Client configuration - config.pm

```
#!/usr/bin/perl

# Peter Jackson November 2005
# Configuration settings for the honeytalk client that reads from a
mysql database
# and periodically sends the data to a honeytalk server.

use constant HT_DB_HOSTNAME   =>  'myhost';
use constant HT_DB_PORT       =>  3506;
use constant HT_DB_DATABASE   =>  'honeypot';
use constant HT_DB_USERNAME   =>  'honeyd_user';
use constant HT_DB_PASSWORD   =>  'bitterlemon';

# The URL of the honeytalk sever to connect to, including a trailing /
# NOTE: if the URL starts with "https://" the HT_HTTPS_* variables
need to be completed
use constant HT_HTTP_BASE_URL =>
'https://server.honeytalk.org/honeytalk/';

# Should connection to honeytalk sever be made using HTTPS
# set to 1 to enable, to disable HTTPS set to 0
use constant HT_HTTPS_ENABLED =>  1;

# a reference to a PKCS #12 encoded client certificate file
use constant HT_HTTPS_PKCS12_FILE => 'client100-signed.p12';

# the password required to unlock the PKCS #12 file, set to empty
# string if no password was specified when creating the PKCS #12 file
use constant HT_HTTPS_PKCS12_PASSWORD => '';

use constant HT_HTTPS_CA_FILE => 'honeytalk-ca.crt';

# the period in seconds between each check for new alerts.
# specifing a low value such as 2 will increase the load
# on the local mysql DB, setting this value to less than the
# "sighist_interval" defined in honeycomb.conf is just a waste
# of time as the data will not change between checks, default to 10
use constant HT_ALERT_CHECK_PERIOD => 10;

return 1;
```

Server script - xmlAlertLog.php

```
<?php
require_once('config.inc.php');
error_reporting(E_ALL);

function handle_error ($msg) {

  echo $msg;
  exit;

}

if (HT_HTTPS_CLIENT_CERTS_ENABLE == true) {
  // check that the client cert was correctly signed by the CA
  if (isset($_SERVER["SSL_CLIENT_I_DN_O"]) &&
    $_SERVER["SSL_CLIENT_I_DN_O"] === HT_HTTPS_CLIENT_CERTS_O &&
    isset($_SERVER["SSL_CLIENT_I_DN_OU"]) &&
    $_SERVER["SSL_CLIENT_I_DN_OU"] === HT_HTTPS_CLIENT_CERTS_OU) {

    // both OU and O match the required values

    if (isset($_SERVER["SSL_CLIENT_S_DN_CN"]) &&
      substr($_SERVER["SSL_CLIENT_S_DN_CN"], 0, 6) === 'client') {

        // the standard format of the CN field is 'clientX' where
        // x is an integer
        $clientId = intval(substr($_SERVER["SSL_CLIENT_S_DN_CN"],6));
    }
  } else {
    handle_error("Error: Client certificate error");
  }
}


// insert the alert into the Database

$db = mysql_connect(HT_DB_HOSTNAME.(HT_DB_PORT != 3306 ?
':'.$HT_DB_PORT: ''),
                    HT_DB_USERNAME,
                    HT_DB_PASSWORD);
if (!$db)
  handle_error('Could not connect to database: ' . mysql_error());

if (!mysql_select_db(HT_DB_DATABASE, $db))
  handle_error('Could not select database');


// Performing SQL query
$query = "INSERT ht_srv_alerts (alert_date, client_id, alert_data) ".
  "VALUES (now(), '".mysql_real_escape_string($clientId)."', '".
  mysql_real_escape_string($HTTP_RAW_POST_DATA)."')";


$result = mysql_query($query, $db) || handle_error('Database INSERT
failed: ' . mysql_error());

$rowsAffected = mysql_affected_rows($db);

if ($rowsAffected !== 1) {
```

```
  handle_error('Database INSERT failed: ' . mysql_error());
}

mysql_close($db);

?>
```

Server configuration - xmlAlertLog.php

```
<?php

// MySQL Connection Constants

define ("HT_DB_HOSTNAME", "localhost");      // the DNS name or IP of
MySQL server
define ("HT_DB_USERNAME", "honeytalk_server");   // the user name for
the MySQL server
define ("HT_DB_PASSWORD", "sweetnector");       // the password for
the MySQL server
define ("HT_DB_DATABASE", "honeypot"); // The name of the MySQL
database
define ("HT_DB_PORT", 3306);         // the TCP port that the MySQL
database is listening on

// if set to true, checks that a valid client certificate has been
used to connect
define ("HT_HTTPS_CLIENT_CERTS_ENABLE", true);

// client certs must match the following O (orgination) and OU
(orginational unit)
// to be permitted access
define ("HT_HTTPS_CLIENT_CERTS_O", "Honeytalk");
define ("HT_HTTPS_CLIENT_CERTS_OU", "Honeytalk Distributed Network");


?>
```

Appendix C - Threat data Document Type Definition (DTD)

honeytalk.dtd

```
<!-- the top level elements alerts can contain any number of alert
elements -->
<!ELEMENT alerts (alert*)>

<!-- alert stores information for each threat -->
<!ELEMENT alert
  (protocol,source?,destination?,date,comment?,flags?,flow?,content?)>


<!ELEMENT protocol (#PCDATA)>
<!ELEMENT date     (#PCDATA)>
<!ELEMENT comment  (#PCDATA)>
<!ELEMENT flags    (#PCDATA)>
<!ELEMENT flow     (#PCDATA)>
<!ELEMENT content  (#PCDATA)>


<!-- source and destination may consist of an address and/or ports -->
<!ELEMENT source (address?,ports?)>
<!ELEMENT destination (address?,ports?)>

<!ELEMENT address (#PCDATA)>

<!-- ports may contain a combination of port and portranges -->
<!ELEMENT ports (port|portrange)*>

<!ELEMENT port (#PCDATA)>

<!ELEMENT portrange EMPTY>
<!ATTLIST portrange           from    CDATA     #REQUIRED
                              to      CDATA     #REQUIRED>
```

Appendix D - Experiment 2 Cisco Device Configuration

Router 1 (Cisco 2600)

```
version 12.2
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Router1
!
memory-size iomem 10
ip subnet-zero
!
call rsvp-sync
!
!
controller T1 1/0
 framing sf
 linecode ami
!
interface FastEthernet0/0
 ip address 10.0.1.254 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface Serial0/0
 no ip address
 shutdown
 no fair-queue
!
interface BRI0/0
 no ip address
 encapsulation hdlc
 shutdown
!

interface FastEthernet0/1
 ip address 10.0.0.254 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
ip classless
ip http server
!
dial-peer cor custom
!
!
line con 0
 exec-timeout 0 0
line aux 0
line vty 0 4
 login
!
end
```

Switch 1 (Cisco 3550)

```
version 12.1
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname switch1
!
ip subnet-zero
!
spanning-tree mode pvst
spanning-tree extend system-id
!
!
interface FastEthernet0/1
 description this interface connects the router upload
 switchport access vlan 2
 switchport mode access
 no ip address
!

interface FastEthernet0/2
 description this is the interface of the SERVER
 switchport access vlan 2
 switchport mode access
 no ip address
!

interface FastEthernet0/3
 description this is the interface of the attack agent
 switchport access vlan 2
 switchport mode access
 no ip address
!

interface FastEthernet0/4
 no ip address
!
interface FastEthernet0/5
 no ip address
!
interface FastEthernet0/6
 no ip address
!
interface FastEthernet0/7
 no ip address
!
interface FastEthernet0/8
 no ip address
!
interface FastEthernet0/9
 no ip address
!
interface FastEthernet0/10
 no ip address
!
interface FastEthernet0/11
 no ip address
```

```
!
interface FastEthernet0/12
 description this is the interface of the sniffer
 no ip address
!
interface FastEthernet0/13
 no ip address
!
interface FastEthernet0/14
 no ip address
!
interface FastEthernet0/15
 no ip address
!
interface FastEthernet0/16
 no ip address
!
interface FastEthernet0/17
 no ip address
!
interface FastEthernet0/18
 no ip address
!
interface FastEthernet0/19
 no ip address
!
interface FastEthernet0/20
 no ip address
!
interface FastEthernet0/21
 no ip address
!
interface FastEthernet0/22
 no ip address
!
interface FastEthernet0/23
 no ip address
!
interface FastEthernet0/24
 no ip address
!
interface GigabitEthernet0/1
 no ip address
!
interface GigabitEthernet0/2
 no ip address
!
interface Vlan1
 no ip address
 shutdown
!
interface Vlan2
 no ip address
!
ip default-gateway 10.0.1.254
ip classless
ip http server
!
!
!
line con 0
```

```
 exec-timeout 0 0
line vty 0 4
 login
line vty 5 15
 login
!
!
monitor session 1 source vlan 1 - 2 rx
monitor session 1 destination interface Fa0/12
end
```

Switch 2 (Cisco 2950)

```
version 12.1
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname switch2
!
ip subnet-zero
!
spanning-tree mode pvst
no spanning-tree optimize bpdu transmission
spanning-tree extend system-id
!
!
interface FastEthernet0/1
!
interface FastEthernet0/2
!
interface FastEthernet0/3
!
interface FastEthernet0/4
!
interface FastEthernet0/5
!
interface FastEthernet0/6
!
interface FastEthernet0/7
!
interface FastEthernet0/8
!
interface FastEthernet0/9
!
interface FastEthernet0/10
!
interface FastEthernet0/11
!
interface FastEthernet0/12
!
interface FastEthernet0/13
!
interface FastEthernet0/14
!
interface FastEthernet0/15
!
interface FastEthernet0/16
!
```

```
interface FastEthernet0/17
!
interface FastEthernet0/18
!
interface FastEthernet0/19
!
interface FastEthernet0/20
!
interface FastEthernet0/21
!
interface FastEthernet0/22
!
interface FastEthernet0/23
!
interface FastEthernet0/24
!
interface Vlan1
 no ip address
 no ip route-cache
 shutdown
!
ip http server
!
line con 0
line vty 5 15
!
!
end
```

Appendix E - Diary Log

16th March 2005

Agreed with Bill that "Detection of Network Threats using Honeypots" topic was suitable for an honours project.

30th March 2005

Perform preliminary experiments using the Sebek kernel data capture tool. My first attempt at building the kernel module failed, the module loaded but no logging packets where observed leaving the host. After some more patching of linux kernel sources and general hacking, I was able to load the module and observe logging packets.

I'm not sure how or even if the high interaction host based honey pots area has something I can develop into a viable honours project. I am leaning toward a lower interaction system such as honeyd.

5th May 2005

Requested an inter-library loan for three articles from IEEE security and privacy journal.

16th May 2005

Had a meeting with Jamie to discuss the contents of my interim report. He was

Received the articles from IEEE security and privacy and quickly had a read of them. They provide good background to the operation of high interaction honeypots operated and details of the data analysis process.

9th June 2005

Project review with Bill and Peter Ross. I was faced with some searching questions that made me realise that I needed to change my focus to something more novel that has potential to provide added value.

I'm going to continue evaluating the various techniques and tools and continue to research the background so I can find something that meets therequirements of an honours project.

I am going to continue looking at low interaction honeypots and possibly combine distributed honeypot aspects.

29th June 2005

19th July 2005

Successfully complied honeyd-0.8 with honeycomb-0.6 on a Debian Linux machine at home. For some reason honeycomb did not want to build with a more recent version (1.0a) of honeyd, some weird compiler error. I'm guessing the cause of the problem is some internal change in honeyd has meant honeycomb no longer can find what it is looking for. I am not about to jump in a do some C hacking, as it's no great problem sticking with honeyd-0.8. I would have licked to have been able to use some of the new features and apparently there are lots of bugfixes.

Did some simple tests with honeycomb including sending the a Apache SSL exploit to a real Apache sever using the proxy functionality.

21st October 2005

Meeting with Jamie, Lionel and Bill. I gave an update of my progress so far and discussed how I might evaluate my implementation.

2nd December 2005

I presented an overview of my project and the associated subject area to the Distributed Systems and Mobile Agents Research Group.

5th December 2005

Meeting with Bill, Jamie and Lionel to discuss the areas of the report that I need to work on further.

Finished evaluation experiments and analysed results.