PROJECT ASSIGNMENT

Testing and validation of a dynamic honeypot system

Department of Telematics
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

Jan Tore Sørensen

December 20, 2006

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROJECT ASSIGNMENT

Student's name:	Jan Tore Søerensen
Course:	TTM4705
Tittle:	**Testing and validation of a dynamic honeypot-system**

Text:

In the spring of 2005, as a part of a master thesis, a student developed a system for *dynamic deployment of Honeypots* . This project consists of configuring and setting a system as described in the master thesis and to perform additional testing and validation beyond that which was done in the thesis. The student should emphasize the importance of running real exploits against the web server. The student should also identify areas of improvement beyond what is described in the chapter "further work".

Deadline:	December 20, 2006
Handed in:	December 20, 2006
Carried out at:	Department of Telematics
Supervisor:	Martin Gilje Jaatun, SINTEF IKT

Trondheim, December 20, 2006

Svein Johan Knapskog
Professor

Abstract

In this assignment we have recreated the system designed by Nyre in [Nyr05]. We have documented the setup procedure and implemented several improvements to the system. By implementing the application servers diskless boot procedure through Pre-boot Execution Environment (PXE), we have eliminated the need for third party programs in the boot process. We have performed testing and validation of the system functions and discovered some weaknesses in the setup, architecture and the data obfuscation function. Through the test phase we discovered that due to different implementations of Hyper-Text Transfer Protocol (HTTP) in Apache 1.3 and Apache 2 the system voter does not support refreshing of web pages. We have installed a centralized log server, yule, to coordinate alerts and provide off-site storage for these log messages. As there were no data control functionality for the honeypot, we have implemented this through IPTables and verified that it blocks all undesirable traffic. We further propose and discuss possible improvements to the proxy, system security and architecture.

Preface

The work on this project has been a learning experience on many levels. I especially appreciated the practical approach of this project and the many challenges it provided. The process of rebuilding this system from scratch, have enabled me to experience the design and implementation phase of a project. There are several people that in various ways have contributed to the process of writing this project and the outcome of it.

First I would like to thank my supervisor, Martin Gilje Jaatun, for proposing this project and for sharing his time, guidance and expertise throughout this work. His enthusiasm, feedback and support have been of great importance to this project.

I would also like to thank SINTEF IKT for providing an office, workspace and the necessary equipment for this project and for including me socially in their working environment.

Professor Svein Johan Knapskog has also contributed to the outcome of this project, in particular in the design and sturcture of the report.

Trondheim, December 20, 2006

Jan Tore Sørensen

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 System Intent . . . 2
- 1.3 Problem description . . . 2
- 1.4 Research Methodology . . . 2
- 1.5 Complicating factors . . . 3
- 1.6 Outline . . . 4

2 Background **5**
- 2.1 Definitions . . . 5
  - 2.1.1 Security . . . 5
  - 2.1.2 Dependability . . . 6
  - 2.1.3 Survivability . . . 7
- 2.2 Honeypots . . . 7
  - 2.2.1 Overview . . . 8
  - 2.2.2 Identifying an attack . . . 9
  - 2.2.3 Data control . . . 10
  - 2.2.4 Data Capture . . . 10
  - 2.2.5 MPITS . . . 11
  - 2.2.6 SAMHAIN . . . 13
- 2.3 Related work . . . 14

3 The system **19**
- 3.1 System Overview . . . 19
- 3.2 Hardware . . . 21
- 3.3 Operating systems, programs and protocols . . . 21
  - 3.3.1 Free BSD . . . 22
  - 3.3.2 Debian . . . 22
  - 3.3.3 Lessdisks . . . 22
  - 3.3.4 DHCP . . . 23
  - 3.3.5 UDHCP . . . 23
  - 3.3.6 PXE . . . 23

3.3.7 TFTP . . . 23
3.3.8 NFS . . . 24
3.4 The file- and application server relationship . . . 24
3.5 Set up . . . 24
3.5.1 Debian File server . . . 26
3.5.2 Fileserver Free BSD . . . 28
3.5.3 Application servers . . . 29
3.5.4 Minimal Proxy for Intrusion Tolerant Systems (MPITS) . . . 29
3.6 SAMHAIN . . . 30
3.7 Problems and differences . . . 30
3.7.1 Data obfuscation Function . . . 31

4 Testing **33**
4.1 Overview, methodology and test technique . . . 33
4.2 Blackbox vs whitebox testing . . . 34
4.3 Tests . . . 34
4.3.1 Test of basic web service . . . 35
4.3.2 Session support in MPITS . . . 36
4.3.3 Testing SAMHAINs file integrity check . . . 36
4.3.4 Testing the data obfuscation functionality . . . 37
4.3.5 Testing IPTables . . . 37

5 Improvements and further work **39**
5.1 Overview . . . 39
5.2 Implemented improvements . . . 39
5.2.1 The initial boot setup . . . 40
5.2.2 Data control . . . 40
5.2.3 SAMHAIN Logging server . . . 43
5.3 Suggested improvements to MPITS . . . 43
5.3.1 Sessions . . . 43
5.3.2 The control network should use broadcast . . . 44
5.3.3 MPITS does not sanitize input or output . . . 44
5.4 Suggested improvements for securing the system . . . 45
5.4.1 Further improvements of the SAMHAIN log server . . . 45
5.4.2 Protection of log traffic . . . 45
5.4.3 Containment of compromised application servers . . . 46
5.4.4 Detection of changes in system state . . . 47
5.4.5 Increasing system surveillance/monitoring . . . 47
5.4.6 A dynamic Honey pot system . . . 47
5.5 Proposals for improving the system architecture . . . 48
5.5.1 Number of entities . . . 48
5.5.2 A new architecture . . . 49
5.5.3 Production environment with use of sticky honeypots / tarpits . . . 51
5.5.4 The data obfuscation function . . . 52

5.5.5 Removing third party software . . . 52
5.5.6 The use of cots vs custom equipment . . . 52

6 Discussion and Conclusion **53**
6.1 System evaluation . . . 53
6.2 Conclusion . . . 54

7 Web resources **58**

A System setup walk-through **61**
A.1 The setup of the Debian based file servers . . . 61
A.2 The setup of the Lessdisks Debian based Application server . . . 65
A.2.1 Setup of Apache 1.3 . . . 65
A.2.2 Setup of Apache 2 . . . 66
A.3 The setup of the FreeBSD based file servers . . . 66
A.4 Setup of the Free BSD based Application server . . . 69
A.5 Setup of the SAMHAIN client . . . 69
A.6 Setup of the MPITS . . . 70
A.7 Setup of SAMHAIN log server . . . 71

B Configuration files **72**
B.1 The configuration files for the Debian fileserver setup . . . 72
B.1.1 /etc/network/interfaces . . . 72
B.1.2 Lessdisk configuration values . . . 73
B.1.3 DHCPD.conf . . . 74
B.1.4 70_DHCP . . . 75
B.1.5 The samhainrc file . . . 77
B.2 THe configuration files for the Lessdisks Debian based Application server 94
B.2.1 etc/network/interfaces . . . 94
B.3 The configuration files for the FreeBSD setup . . . 95
B.3.1 etc/rc.conf . . . 95
B.3.2 /usr/local/etc/dhcpd.conf . . . 96
B.3.3 /usr/src/sys/i386/conf/APPSERVER . . . 97
B.4 The configuration files for the Free BSD Application server setup . . . 102
B.4.1 /etc/rc.conf . . . 102
B.5 The configuratio files for the MPITS . . . 103
B.5.1 /etc/network/interfaces . . . 103
B.6 SAMHAIN messages . . . 104
B.6.1 The initialization message from SAMHAIN . . . 104
B.6.2 The alert message from SAMHAIN . . . 105

List of Figures

2.1 Laprie's Taxonomy of dependability [Lap95] 6
2.2 Overview of the process of deciding the appropriate response to the user. The rectangular boxes indicate the proper response (above) and additional operations to be executed (below)[Nyr05]. 13
2.3 Schematic view of Deswartes et al intrusion-tolerant server architecture [VAC+03]. 15
2.4 A generic content agreement protocol proposed by Deswart et al in [VAC+03] 16
2.5 An overview of the system architecture for a dynamic content web service. [SDN03] 17
2.6 An overview of the dynamic web server mode of operation [SDN03]. The abbreviation RID is short for request identity. 18

3.1 An overview of the system architecture 20
3.2 The communication between file and application server during boot . . . 25

5.1 An overview of the system architecture with IPTables. IPTables is running on the interface above each host and on both payload data interfaces on the MPITS 42

List of Tables

3.1 The network configuration for each Network Interface Card (NIC) in the system architecture. The X is substituted with the server unit number to obtain the full addresses. 19
3.2 The hardware setup of the different server types 21
3.3 The OS distributions and web server versions of the application servers . 21

5.1 Table of open ports on each network interface. These are the only open ports on the different system types. The blocking is done by an IPTables script. Cl refer to client and sr to server. 41

Acronyms

BIOS Basic Input/Output System BIOS is installed on the computer's motherboard. It controls the most basic operations and is responsible for starting your computer up and initializing the hardware. It is data that is usually held in a ROM chip, which can be updated by "flashing". BIOS upgrades may correct errors, support new CPUs, support new hardware, etc.

BOOTP Bootstrap Protocol BOOTP is an earlier IETF-defined booting protocol that is much less flexible than DHCP.

COTS Commercial off-the-shelf COTS is an expression used to describe standard software packages available through retail channels.

DHCP Dynamic Host Configuration Protocol DHCP is a protocol for assigning dynamic IP addresses to devices on a network.

FTP File Transfer Protocol FTP is one of the protocol used on the Internet for exchanging files. FTP uses the Internet's TCP/IP protocols to enable data transfer.

HVAC Heating, Ventilating, and Air Conditioning HVAC refers to a system installed in a computer room to regulate temperature. This includes air conditioning plants, chillers and ducting systems, which ensure the uniform transfer of the cold or hot air, as the case may be throughout the computer room.

HTTP HyperText Transfer Protocol HTTP is the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

IDS Intrusion Detection System IDS monitors any network traffic and logs any possible malicious activity. Unlike a standard Firewall, IDS can differentiate between friendly and unfriendly activity.

IP Internet Protocol IP is a standardized method of transporting information across the Internet in packets of data. It specifies the format of packets, also called datagrams, and the addressing scheme.

LAN Local Area Network LAN is a computer network covering a local area, like a home, office, or group of buildings.

MPITS Minimal Proxy for Intrusion Tolerant Systems MPITS is the proxy described in [Bro05], which is used in our system.

NFS Network File System NFS is a network service that lets a program running on one computer use data stored on a different computer on the same network as if it were on its own disk.

NIC Network Interface Card NIC is a computer circuit board or card that is installed in a computer so that it can be connected to a network. Network interface cards provide a dedicated, full-time connection to a network.

OS Operating System OS is the set of basic programs and utilities that make your computer run and enables it to run other programs. The operating system perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

PXE Pre-boot Execution Environment PXE is short for Pre-boot Execution Environment, which allows booting a client off a network server, using the NICs ROM.

PGP Pretty Good Privacy PGP is an electronic privacy program which helps you ensure privacy by letting you encrypt files and e-mail. The encryption technology employed by PGP is very strong. PGP was created by Phil Zimmermann, and depends on public key cryptography for its effectiveness. Public key cryptography is a procedure in which users exchange "keys" to send secure documents to each other.

RAM Random Access Memory RAM the main memory of a computer system used for storing programs and data. RAM provides temporary read/write storage.

RPC Remote Procedure Calls RPC enables a system to make calls to programs such as NFS across the network transparently, enabling each system to interpret the calls as if they were local. In this case, it would make exported file systems appear as thought they were local.

SPOF Single Point of Failure SPOF is a service or component that, if it fails, will cause the entire system to go down.

SSL Secure Sockets Layer SSL is used by most commerce servers on the World Wide Web. It is high-level security protocol that protects the confidentiality and security of data while it is being transmitted through the Internet. It is based on RSA Data Security's public-key cryptography, SSL is an open protocol that has been submitted to several industry groups as the industry security standard.

TCP Transmission Control Protocol TCP is a IP based protocol to send data over the Internet. TCP is a connection-oriented protocol, which means that a connection is established and maintained until such time as the message or messages to be exchanged by the application programs at each end have been exchanged. TCP is responsible for ensuring that a message is divided into the packets that IP manages and for reassembling the packets back into the complete message at the other end.

TFTP Trivial File Transfer Protocol TFTP is a very simple file transfer protocol akin to a basic version of FTP. TFTP is often used to transfer small files between hosts on a network, such as when a remote X Window System terminal (or any other thin client) boots from a network host or server.

TLS Transport Layer Security TLS is a function providing encryption and authentication services that can be negotiated during the start-up phase of many Internet protocols (e.g. SMTP, LDAP, IMAP, POP3). TLS is derived from SSL and uses the same certificates but does not require each service to be given a new port number.

UDHCP UDHCP UDHCP The UDHCP client negotiates a lease with the DHCP server and executes a script when it is obtained or lost.

UDP User Datagram Protocol UDP is a communications protocol for the Internet transport layer, which makes it possible to send a datagram message from one computer to an application running in another computer. Like TCP (Transmission Control Protocol), UDP is used with IP (the Internet Protocol). Unlike TCP, UDP is connectionless and does not guarantee reliable communication; the application itself must process any errors and check for reliable delivery.

Chapter 1

Introduction

In this chapter we will present the super eminent objective of our work and the underlying motivation for this assignment.

1.1 Motivation

The Internet or the World Wide Web (WWW) has significantly impacted every facet of daily life in many parts of the world. The user-base has grown exponentially since the 90s. More and more people get familiar with the web every day and integrate it in their daily routine. As society is growing increasingly dependent upon the services offered on the WWW, the more important it is for the service providers to actually be able to deliver the requested services despite potential malicious users interested in shutting the service down. The task of providing a trivial web service is dependent on complex back end systems. Security measures play an important role in the ability to provide these services.

In traditional security thinking, we wish to create a fortress with inner and outer walls, moats and ramparts to keep intruders from entering. But the security world is constantly changing as new exploits and security holes emerge. Despite the best efforts of security practitioners, no amount of hardening can assure that a system that is connected to a network will be invulnerable to attack. To counter this we aim for system that use redundancy and diversity to survive. The concept of survivability introduces us to systems that can deliver essential services and maintain essential properties such as integrity, confidentiality, and performance, despite the presence of intrusions. Therefore to defend your system you must know who your enemy is; their methods of attack, tools, objective and tactic. The purpose of scientific application of honey pots is to learn from the attackers without the attackers doing any real damage to important systems. In this project we will look closer into one such system design to test its use as a deployed system.

1.2 System Intent

The system we will be looking into was implemented in [Bro05] and [Nyr05]. The system aims to be able to continue to deliver services despite of being attacked. It is based on the idea that redundancy and diversification in components can increase the survivability of a system by only using Commercial off-the-shelf (COTS) instead of utilizing special hardware. The system is meant as a proof of concept which aims to prove that by using a diverse range of operating systems and web servers the system is better able to withstand attacks based on system-specific exploits. To detect exploits, the system should use a method of detection that is so general that it also includes unknown exploits. Our system has two means of detection. It runs a voting mechanism on a front end proxy, which act as an intermediary between the client and our service. The proxy compares responses from a redundant and diversified set of web servers and forwards the majority response from the back end system. The second mechanism detects unwanted changes in the system integrity, and interprets such a change as an intrusion. The attacker should be able to retain control of the compromised component without revealing any sensitive information or allowing the attacker to conduct further attacks as proposed by Jaatun and Hallingstad in [JH02]. By having a redundant two-level detection mechanism, we hope to create an intrusion-tolerant system. If a component is successfully attacked, the system obfuscates valuable data on attacked systems so that the attackers do not gain any important information from the system. At the same time as the compromised system is detected it will start functioning as a honeypot and gather information from the attack, but this area is not explored by [Nyr05].

1.3 Problem description

We will reconstruct the system as described in [Nyr05] and document the setup of the system. We will use the same software as in [Nyr05] to recreate its setup. When the system is set up, we will conduct extensive testing on the system. Some of the tests will be preformed by running actual exploits against the system to document that it is functioning the way it should. During the test period we will try to identify weaknesses and areas of improvement and evaluate the value of such a system design.

1.4 Research Methodology

The working title of this project is *Testing and validation of a dynamic honeypot-system*, and in an attempt to formalize our work, we feel that we must include a brief discussion of research methodology in our work. According to Merriam-Webster's thesaurus [1] research is "a systematic search for the truth or facts about something". We feel that the scientific methodology of classical research where we define a hypothesis and then

conducts tests to prove/support our hypothesis does not apply to our technological specialization field. In [Gla11] they state that "software research is in a sad state" and that "without a scientific method, techno babble, lemminginerring, and fads run rampant". To take a more scientific approach, we therefore turn to [SSl06] to define our work. There they define technology research as "scientific technology involving the production of new or improved devices especially in the fields of electronics and computers". We feel that this is a much more adequate definition of our work. [Gla11] presents possible four models for use in scientific research:

1. **The scientific method** Observer the world, propose a model or theory of behavior, measure and analyze, validate hypotheses of the model or theory and if possible repeat.

2. **The engineering method** Observe existing solutions, propose better solutions, build or develop, measure and analyze, repeat until no further improvements are possible.

3. **The empirical method** Propose a model, develop statistical or other methods, apply to case studies, measure and analyze, validate the model, repeat.

4. **The analytical method** Propose a formal theory or set of axioms, develop a theory, derive results, and if possible compare with empirical observations.

In our work we have mainly followed the engineering method, but there may be hybrid elements from the other models involved in our work. We have not focused on finding quantifiable results in the project, but rather use qualitative means to assess the system in question. When it comes to the testing and validation part of the project we have not followed any specific testing and validation technique but rather conducted experiments focused on verifying the functionality described in [Nyr05]. [ZW05] deals with experimental models for validating technologies but we feel that applying such models is beyond the scope of this project.

1.5 Complicating factors

When writing this project we met several unforeseen problems. During the setup phase we discovered that [Nyr05] contained very little documentation on the actual setup of the underlying services and the programs used. The documentation provided there did not work on the hardware we acquire for this project. The Lessdisks program used by [Nyr05] is no longer available at sourceforge and the project appears to be closed down. There is little documentation available at the Lessdisks homepage [2] and the program was never released as a stable 1.x version. Due to this we spent much more time on setting up and configuring the system correctly. Thus, we had less time to test and run exploits on the system than first anticipated. To save others from re-inventing the wheel and redoing this work, we have created a thorough documentation of the set up and the

problems we encountered during this phase.

1.6 Outline

The remaining parts of this project are organized as follows. In chapter two we give an introduction to important terms, concepts, theory and software used in our setup. We also present a similar system by Valdes et al, described in [VAC+03]. In chapter three we explain techniques, functionality and software used in our system and provide a complete overview of the setup and configuration phase of our system. Chapter three together with appendix A documents the entire setup procedure. Appendix A should be read as a supplement to chapter three as it is a complete walkthrough of the system setup and will help the reader gain insight into system specifics. First part of chapter four deals with testing in a theoretical manner, followed by our tests and results. In chapter five we identify areas of improvement and discuss our proposals using theory from chapter 2 to support our view. Chapter five is also perceived as our further work chapter as we identify areas of improvement here. In chapter six we evaluate our system against that of [VAC+03] and present our concluding remarks.

We have distinguished printed references from their web-based counterparts to clarify the origin of these resources. The published references are cited in alphabetic order using the abbreviation of the first author's last name and publishing year (e.g. [DeL06]), while the web resources uses an numeric citation style (e.g. [1]).

Additionally, we emphasize that the usage of first person plural (i.e. *we*), is only due to common convention and is therefore to be interpreted as *the author*.

Chapter 2

Background

In this chapter we will look into the theory behind the Dynamic Honey pot System, give definitions, introduce and explain important concepts and look at a similar system designed by Valdes et al [SDN03].

2.1 Definitions

Before we can look into the internal workings of the Dynamic Honey pot System, it is important to have a common understanding of the concepts and technical terms behind the system. Since this project is based on [Nyr05] and [Bro05], we will strive to be consistent with the definitions used in these theses. Although we will provide brief definitions of security, dependability and survivability , we will not attempt to repeat the detailed discussion of theory in these earlier efforts.

2.1.1 Security

To define security we will use the definition given by Shirey in RFC 2828 [Shi00]:

1. Measures taken to protect a system.
2. The condition of a system that results from the establishment and maintenance of measures to protect the system.
3. The condition of a system‘s resources being free from unauthorized access and from unauthorized or accidental change, destruction, or loss.

To be consistent with the earlier work of Nyre in [Nyr05] we will also include the definition from Laprie[ALRL04]. He defines security as the “composite of the attributes of confidentiality, integrity, and availability”. We feel that, even though confidentiality is

not critical for our web server, the attributes of integrity and availability are. So we will use the definition of Laprie as it is shorter and more to the point. We can of course picture other areas of application which would require all attributes, but those are beyond the scope of this project.

2.1.2 Dependability

The dependability of a system is defined by Helvik in [Hel03] as *the trustworthiness of a system such that reliance can justifiably be placed on the service it delivers.* Laprie has a definition in [ALRL04] which we find more suitable to our system. He defines dependability as *the ability of a system to avoid service failures that are more frequent or more severe than is acceptable.* This is the definition we will be using as we feel it embraces the nature of our system. Dependable systems often rely on redundancy and diversification to achieve high levels of system availability and reliability. We wish to use Laprie's Taxonomy of dependability to formalize the concepts later used in this in this project. As we can see in in figure 2.1.2, Laprie classifies the concepts of dependability in 3 attribute classes.

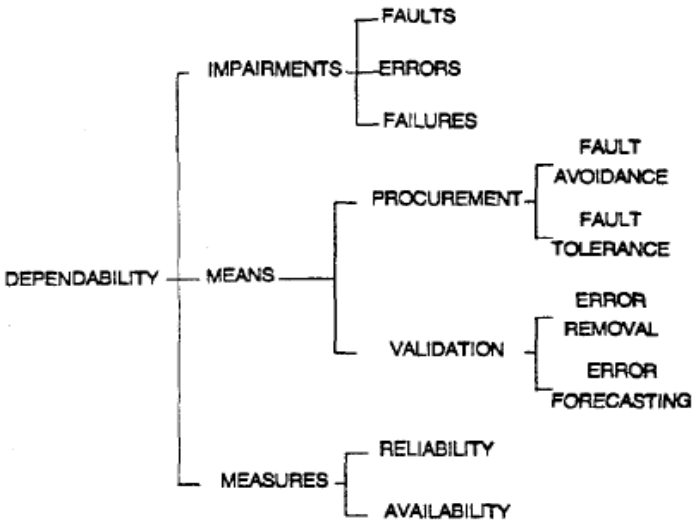

Figure 2.1: Laprie's Taxonomy of dependability [Lap95]

The possible impairments that can effect a system's dependability are classified under impairments. We will not dwell further on these as they all have the same degrading effect on our system.

The means to achieve a dependable computing system include a set of methods that can be classified into four categories[Lap95]:

- **Fault-avoidance:** means to prevent the occurrence or introduction of faults.

- **Fault-tolerance:** means to avoid service failures in the presence of faults.
- **error-removal:** means to reduce the number and severity of faults.
- **error-forecasting:** means to estimate the present number, the future incidence and the likely consequences of faults.

Some of these means are used as functions in our system and will be discussed later. To survey a system's dependability one uses the measures of reliability and availability. In this project we wish to focus on the system functions and not directly measure the system's dependability.

2.1.3 Survivability

Survivability has to be defined in the correct context. The old term of survivability was a measure of how many nodes the enemy had to destroy before bringing the network down. We view all threats that are potentially damaging to the system as incidents. We wish to use survivability as described in [EM99] by Ellison et al. to describe the systems ability to resist such incidents. In [EM99] Ellison et al. defines survivability as *the capability of a system to full fill its mission, in a timely manner, in the presence of attacks, failures, or accidents.* An attack is a potentially damaging event orchestrated by an intelligent adversary whereas failures are incidents caused by deficiencies inside the system and accidents are incidents with a random external cause.

We observe that Ellison et al. focus on the impact of the incident rather the the cause, because of the difficulty in separating an attack from a random system failure.

2.2 Honeypots

The term honeypot or honeytrap was used during the cold war as a name for employing sexual entrapment to gain information from an enemy. In computer terminology, a honey pot is a system set up to attract and be compromised by attackers for the purpose of deflecting them from real systems and learning from the attackers. The concept of using honeypots is not a new concept. In the book The Cuckoo's Egg[Sto89] we follow Cliff Stoll's hunt for a hacker using honey pot like methods. He posted fake data he knew the hacker would find interesting to keep the hacker occupied in his system while he was tracing him.

RFC 2828 defines a honeypot as "A system (e.g., a web server) or a system resource (e.g., a file on a server), that is designed to be attractive to potential crackers and intruders, like honey is attractive to bears." The RFC also refers to entrapment under honeypots, and defines entrapment as "The deliberate planting of apparent flaws in a system for the purpose of detecting attempted penetrations or confusing an intruder about which flaws to exploit."[Shi00]. This definition seems somewhat tedious so we prefer the shorter

definition of Spitzner in [Spi03] where he defines honeypots as a "security resource who's value lies in being probed, attacked or compromised".

2.2.1 Overview

It might seem strange, and a waste of resources to dedicate a host to a honeypot, but in a normal computer network it can be very difficult to separate legitimate from malicious traffic. To be able to detect an intrusion you must run some sort of surveillance on your network. Such Intrusion Detection System (IDS) generate a lot of false positives and can keep system administrators constantly on their toes. The honeypot solves this problem by stating that any traffic on the honeypot is malicious. So if someone initiates a connection to a honeypot it is very likely that it is an attempt at scanning or probing of the system. The honeypot should emulate a real system with real users, but only run fake services, not contain any real data or users. The services seem real enough to attract an attackers interest by responding correctly to requests and probing.

It is common to categorize honeypots by their characteristics. A high-interaction honeypot can be compromised completely, allowing an adversary to gain full access to the system and use it to launch further network attacks. Low-interaction honeypots simulate only services that cannot be exploited to get complete access to the honeypot. Low-interaction honeypots are more limited, but they are useful for gathering information at a higher level; for instance to learn about network probes or worm activity. They can also be used to analyze spammers or for active countermeasures against worms.

In [Spi03] Spitzner divides honeypots into two categories; research and production.

Research Honeypots

Research honeypots have a primary purpose of gathering information on attackers[Spi03]. The goal of a research honeypot is to provide intelligence on the intruder‘s behavior on the system and the methods and means utilized to gain access. Even their keystrokes and tools can be analyzed by the researchers. This is valuable information for researchers, as it can help them to identify weaknesses in systems and make them known to the public. It can also give researchers information about new tools, methods and exploits used by attackers.

Production Honeypots

The purpose of production honeypots is to protect the organization; they directly increase resource security. A production honeypot can be deployed to prevent, detect or respond to attacks on a real system. It can prevent an attack by slowing down automated attackers by posing as real hosts. Such honeypots are called Sticky Honeypots or tarpits.

The concept behind a tarpit is fairly simple. We accept incoming connections, but we don't allow them to get back out. When the data transfer begins, the Transmission Control Protocol (TCP) window size is set to zero, so no data can be transferred within this session. The connection is then held open, and any requests by the remote side to close the session are ignored. This has some interesting effects because the attacker must wait for the connection to timeout in order to disconnect. This will consume a bit of the attackers resources. This kind of behavior could also be bad news for automated scanning tools, for instance worms, because they rely on a quick turnaround from their potential victims.

One such system is Labrea[3]. It is a "no-interaction" honeypot designed to deal with worms and network scanners. It takes over unused Internet Protocol (IP) addresses and creates virtual hosts that answer to connection attempts in a way that causes the machine at the other end to "get stuck", sometimes for a very long time. By giving the attackers multiple hosts it can prevent real systems from being attacked.

2.2.2 Identifying an attack

To date, it seems unlikely that we ever will be able to completely prevent security breaches. Therefore we must try to detect these intrusions as they occur and take action to prevent the attackers from doing further damage. There are many ways of detecting and classifying attacks. Due to the enormous amount and diversity of attacks this is clearly a task for a computer. The standard way of detecting attacks today is by using an Intrusion Detection System (IDS). IDS monitor network traffic to determine if an intrusion has occurred. The two basic methods of detection are signature- and anomaly-based detection[Den87]. All network traffic is scanned by the IDS looking for specific features or patterns that might indicate an attack or intrusion. Signature based methods, also known as misuse detection[DeL06], looks for a specific signature to match, which would signal an intrusion. A misuseIDS uses a database of traffic and activity patterns related to known attacks to identify and categorize malicious activity on the network. This is somewhat similar to virus detection since it can only can detect known patterns and thus leave the system open for "zero day exploits". The default setup of Snort[4] uses signature-based detection.

Another approach to intrusion detection is called anomaly detection. Anomaly-based systems attempt to map events to the point where they "learn" what is normal and then detect behavior that diverge from this norm. Anomaly detection techniques assume that all intrusive activities are necessarily anomalous. The greatest challenge in such systems is to select thresholds levels that avoid false positives. In [DeL06] DeLooze proposes to use Self-organizing maps and unsupervised learning techniques to categorize attacks. Her results shows that she generated a very low amount of false positives using this technique with a overall detection rate of 97.31 % [DeL06].

A more general strategy is to detect unwanted changes in the system state. The idea is

that intruders, in order to perform some action on the system, will change the system state by leaving traces of their actions. By monitoring key parameters such as binary signatures and size of files that should not change a program can detect intrusions through changes in data integrity. The program generates a database with signatures of important files using some hash-function and detect discrepancy between the actual file and the database signature. In most cases this requires the database to be located on a physically different location.

2.2.3 Data control

There is always a risk of an attacker or malicious code using a honeypot to launch an attack on another system, or abusing the honeypot in some unexpected way. So when a honeypot becomes compromised, we have to contain the activity to ensure the honeypots are not used to harm other systems. This means that we must be able to control the traffic in and out of the honeypot, without the attacker‘s knowledge. The challenge is to allow the attacker enough freedom to keep her occupied, without risking that she circumvents the data control functions and launches attacks against other hosts.

Just like when it comes to secure a system the approach to Data Control does not rely on a single mechanism. Instead, Data Control is implemented using layers, such as counting outbound connections, intrusion prevention gateways, or bandwidth restrictions. The combination of several different mechanisms help protect against a Single Point of Failure (SPOF), especially when dealing with new or unknown attacks. Also, Data Control should operate in a fail closed manner. This means if there is a failure in your mechanisms (a process dying, hard drive full, misconfigured rules) the honey net architecture should block all outbound activity, as opposed to allowing it. Keep in mind, we can only minimize risk. We can never entirely eliminate the potential of an attacker using a honey pot to harm non-honey pot systems.

2.2.4 Data Capture

The goal of data capture is to provide reliable data to reconstruct the attacker's actions, and thus to gain valuable information from the compromised honeypot. If we look back at Spitzner‘s definition of honeypots in section 2.2, he states that the value of a honeypot lies in being probed, attacked or compromised. When it comes to data capture in honeypots there are two important issues: First, for a honeypot to have any scientific value, we will need documentation of the attacker‘s activities within the honeypot. This requires extensive monitoring and logging of the commands used without the attacker detecting these processes. If the attacker suspects that he is dealing with a honeypot, we may lose valuable data as the attacker most likely will alter his objective and use some drastic means of covering his tracks. Almost all attackers will try to hide and remove their electronic footprints from the system in some way. In [Acc99], we can read

about some of the techniques attackers use to cover their tracks. Since we can not be certain what an attacker may detect, alter or erase, the principle of layering applies here as well. We should use a combination of capturing processes on layers to ensure that we capture enough data about the attacker. We need this data to be safe from the attackers tampering. This brings up the second issue: We must slip the data away from the honeypot under the attacker's nose, and store it on a separate and secure system. The most common technique for doing this today is to use Sebek[5] , which is a tool designed for data capture. It attempts to capture most of the attackers activity on the honeypot by using rootkit techniques for hiding and bypassing the network stack to transport data away from the honeypot undetected. One example of a honeypot which uses data capture as a tool is Nepenthes[6]. It emulates known vulnerabilities in order to collect information about new exploits for old vulnerabilities.

2.2.5 MPITS

We use the Minimal Proxy for Intrusion Tolerant Systems (MPITS) described in [Bro05] as the front end of our system. The MPITS system is responsible for one of our two intrusion detection mechanisms, the voter. If the vote is not unanimous, this is an indication that one of the application servers may be compromised. The MPITS is written in C and is available under version 2 of the GNU General Public License and the source code is therefore available as open source. We will give a short summary for the inner workings of the MPITS. For more information we refer you to [Bro05]. The functions of the MPITS in [Nyr05] are threefold, it acts as a logger, proxy and voter. The logger logs information sent on the control data network for use in a post mortem analysis of a compromised application server. Our proxy, the MPITS differs from a regular proxy on some accounts. A regular proxy server will continuously forward requests to application server. MPITS, queries all application servers and stores the responses in memory. After querying for a response at all application servers, the responses are processed by the voter. The voters functionality may be decomposed into the the following steps[Nyr05]:

1. The voter compares the response PDU's (Protocol Data Units) and determines whether they are equal.
2. The voter places the PDU's into equality classes.
3. The voter, based on the equality algorithm, then determines the correct response PDU and sends this to the client.

Since the application servers run different web servers, their responses may differ somewhat due to implementation differences. Therefore it is important that the MPITS allow a deviation within predetermined threshold limits. The MPITS system uses an equality algorithm to determine if two PDUs are sufficiently equal[Bro05]. The algorithm compares binary data byte by byte creating a string representation of each byte in the two PDUs and counts the number of equal bytes (E_B) and unequal bytes (U_B). This gives the relationship

$$d = \frac{E_B}{E_B + U_B} \tag{2.1}$$

where d is called the normalized Hamming distance. This distance expresses the conformity of the two PDUs. The first PDU the MPITS receives is put into an equality class. When the MPITS receives a second response arrives, the normalized Hamming distance d is calculated and if d is below the predefined threshold limit, the two PDUs are considered equal and put into the same equality class, otherwise they are considered unequal and put into different equality classes [Nyr05].

As the integrity of numeric data, e.g HTTP status codes, can be more critical to maintain than textual data, MPITS deals with numeric input a different way. The numeric data are counted and compared separately from the textual data. To check for numerical discrepancies, MPITS scans the two PDUs for numerical values. The MPITS counts the number of equal (E_N) and unequal (U_N) numeric values[Nyr05]. To calculate the equality relationship d of two PDUs with both data types, the MPITS uses a user defined numerical weight w in the following relationship.

$$d = \frac{U_B + U_N \cdot w}{(U_B + E_B) + (U_N + E_N) \cdot w} \tag{2.2}$$

The distance should also here be below a predefined threshold limit for two PDUs to be considered equal. But as Nyre comments in [Nyr05], there is something that does not make sense about the way the algorithm distinguishes between binary and textual data:

> The algorithm handles both binary and textual data, but does not make a distinction and treats all data as textual. It may therefore consider numerical values within a binary PDU, and subsequently weight them differently. This does not make sense but the impact is assumed to be moderate, hence it is not corrected

All PDUs are evaluated against the first PDU in the equality class. This use of the first PDU as a basis of comparison brings up several evaluation problems due to the use of the first PDU as a baseline of comparison. This is further commented in [Nyr05]. A flow diagram for the voting possible outcomes of the voting process is depicted in figure 2.2.5.

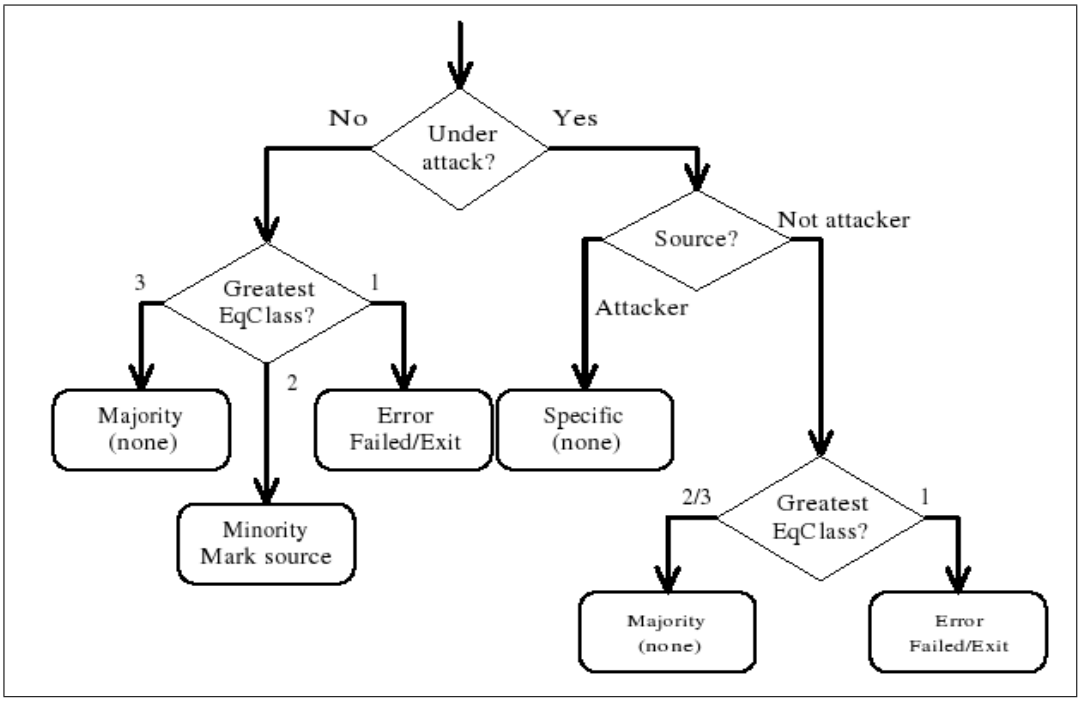

Figure 2.2: Overview of the process of deciding the appropriate response to the user. The rectangular boxes indicate the proper response (above) and additional operations to be executed (below)[Nyr05].

- **Majority:** the response corresponding to the majority decision of the voting unit.
- **Minority:** the response corresponding to the minority decision of the voting unit.
- **Specific:** the response from the attacked server.
- **Error:** the predefined error response.
- **Failed/exit:** the system has failed, shutdown required.
- **Mark source:** the attacker must be logged for future reference.

2.2.6 SAMHAIN

SAMHAIN is the second of our intrusion detection mechanisms and is available under the GNU General Public license. It is a data integrity and intrusion alert system that detects unwanted changes in the system state, as explained in section 2.2.2. SAMHAIN uses a integrity database containing signed hash values for the files and directories it monitors. The starting point and reference value of SAMHAIN is called the baseline[7]. We will use this program to run file integrity checks on important parts of our application servers. The SAMHAIN system comes with a client server option, which we will use.

The log server will receive all log reports from the SAMHAIN clients running on the file servers. The client server setup has several security measures included. It encrypts the communication between the SAMHAIN client and SAMHAIN log server with the Rijndael 192 bits Advanced Encryption Standard (AES) algorithm providing confidential communication. Each SAMHAIN client must be registered at the log server with a hostname, salt and a password to make a connection. At connection to the server the client tells it's host name to The client identifies its self by its host name to the server, and the server verifies this against the peer of the connecting socket. The first time a client connects to the log server, an authentication protocol is performed. This protocol provides mutual authentication of client and server, as well as a establishing a fresh session key. This key expires after two hours. After this the client must repeat repeat the authentication protocol to set up a fresh session key. The log server keeps track of all clients and their session keys. As connections are dropped after successful completion of message delivery, the log server may serve a large number of clients. If the client detect a change in system state it will send a signed message to the log server. On receipt, the log server will do the following steps:

1. Verify the signature.
2. accept the message if the signature is valid, otherwise discard it and issue an error message.
3. Discard the clients signature.
4. Log the message, and the client's hostname, to the console and the log file adding its own signature to the log file entry.

[8]

2.3 Related work

We have found several papers which propose architectures for intrusion tolerant Internet servers. We will not discuss the SITAR and similar projects discussed in [Nyr05] and [Bro05]. Instead we will look at a generic architecture for intrusion tolerant Internet servers proposed by Valdes et al in [VAC+03]. We will especially look into their mechanisms for achieving the attributes of integrity and availability. [VAC+03] aims to build systems that are able to survive attacks in the context of an open network by only using COTS components.

Their design was founded on the following assumptions:

1. No more that a critical number of servers are in an undetected compromised state at any given time.
2. Attackers do not have physical access to the system or configuration.

3. All non-faulty and non-compromised servers generate the same response to the same client request.

The system goal is to deliver the correct service to its legitimate clients, even when attackers are corrupting elements within the system. To achieve this they use some fault tolerant techniques, such as redundancy and diversification, mentioned in 2.1.2.

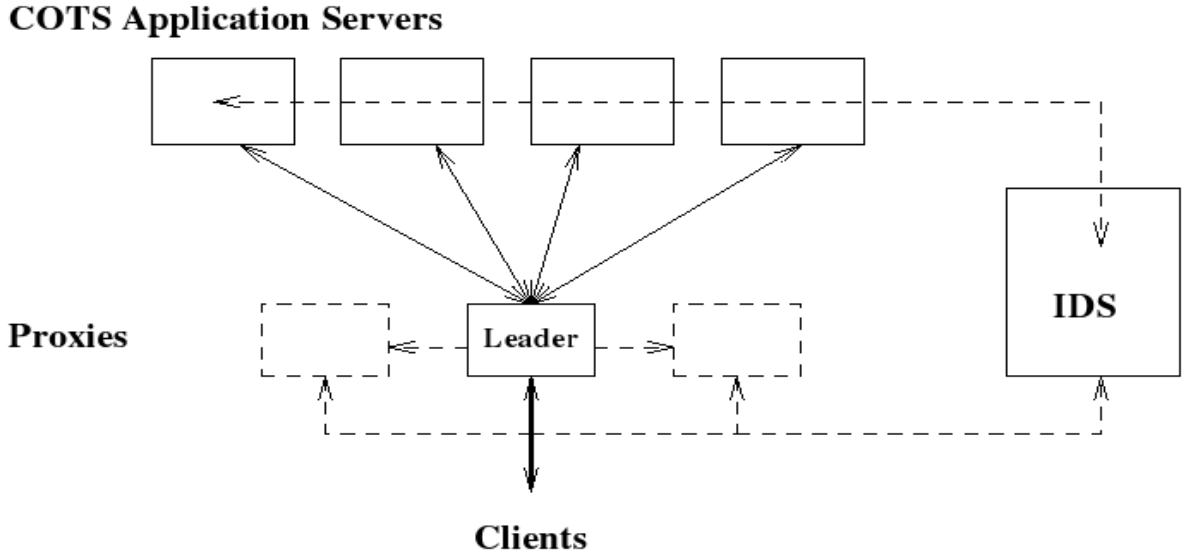

Figure 2.3: Schematic view of Deswartes et al intrusion-tolerant server architecture [VAC+03].

As seen in 2.3, the system uses COTS equipment for the application servers. Each application server provide equivalent services, but is running different web service applications on a diverse selection of operating systems. [VAC+03] states that this should make it unlikely that they are vulnerable to common attacks and failure modes. The only difference between a normal web server and the application server is that the application server runs IDS monitors. All application servers provide the same data content to the proxies.

Valdes et al proposes to use redundant proxies[1] as a front end system. These proxies mediate client requests and use a monitor subsystem to monitor all other entities, including the other proxies, in the system. In [VAC+03] each tolerance proxy can play two different roles: Leader or Auxiliary proxy.

The leader is responsible for filtering and sanitizing client requests as well as load balancing. The leader will then forward these requests to one or more application servers, in accordance with with a *content agreement protocol*. The other proxies, the auxiliary proxies monitor all communication between the leader and the application servers. In

[1]These proxies are denoted as Tolerance proxies in [VAC+03]

addition the IDS depicted in figure 2.3 monitor all system entities. If a majority of the auxiliary proxies or the IDS detects a leader failure, the auxiliary proxies will select a new leader by running an election protocol. In [VAC+03] the auxiliary proxies are referred to as optional and their design and implementation focus on a single proxy, which is similar to our MPITS.

The *content agreement protocol* and number of application servers involved in answering a client request is decided by the proxy leader based on the current *regime*. In [VAC+03], the *regime* specifies, at each point in time, the following parameters:

- Which application servers to forward the request to.
- What constitutes a sufficient agreement among the replies.
- Which servers, if any, to report as suspicious to the monitoring subsystem.

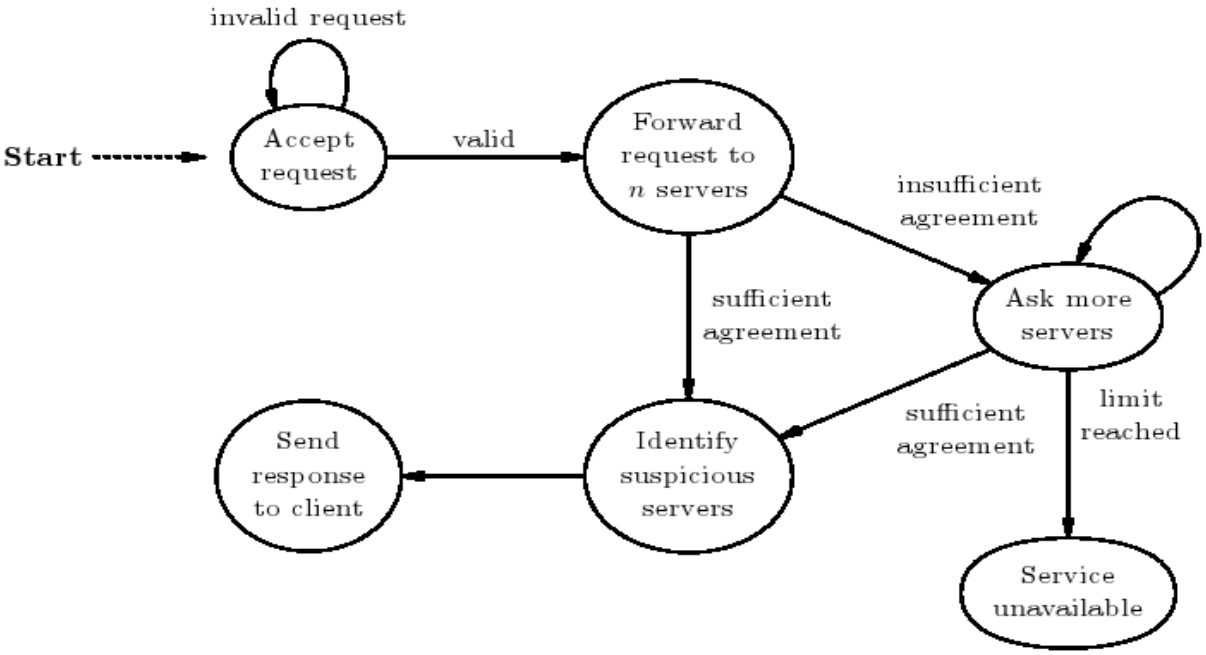

Figure 2.4: A generic content agreement protocol proposed by Deswart et al in [VAC+03]

The third component, the IDS is part of the monitoring subsystem. This is a diversified set of complementary mechanisms for collecting system information. This information is aggregated into a global system view which is used to adapt the regime to respond to suspected or verified threats.

[VAC+03] is a very comprehensive article which explains many of details of their architecture and system. But we will just quote [VAC+03] on their functional overview of their system. We will use this as a basis of comparison when we evaluate our system.

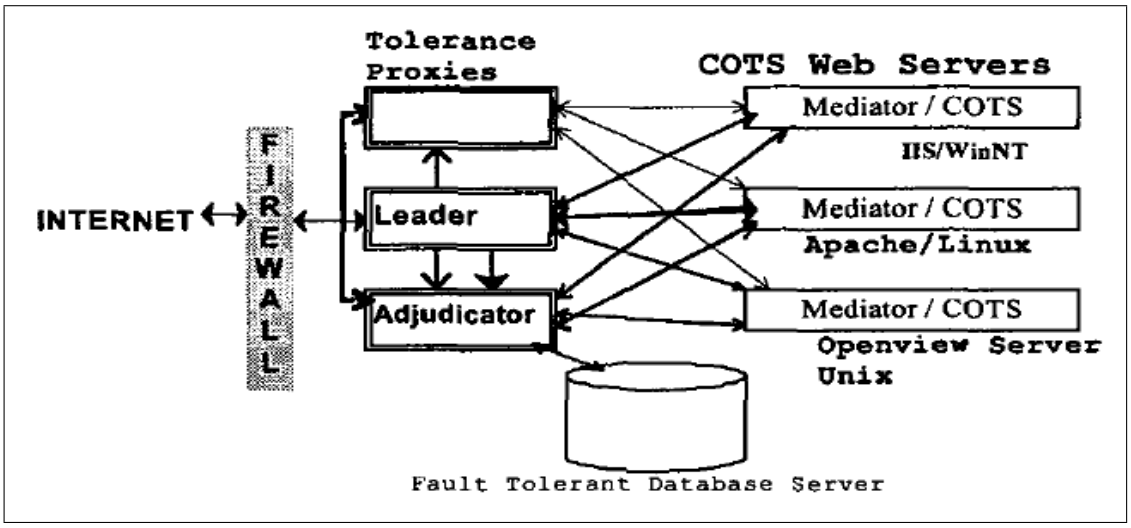

Figure 2.5: An overview of the system architecture for a dynamic content web service. [SDN03]

When a client request arrives, the following steps are performed:

1. *The proxy leader accepts the request and checks it, filtering out malformed requests.*
2. *The leader forwards the request, if valid, to a number of application server, depending on the current regime.*
3. *The application servers process the request and return the results to the proxy leader. If sufficient agreement is reached, the proxy forwards the content to the client.*
4. *The regime is adjusted according to the result of the content agreement protocol and reports from the monitoring subsystem.*
5. *The auxiliary proxies , if present, monitor the transactions to ensure proxy leader behavior.*

[VAC+03]

Saidane et al in [SDN03] is a continuance of the work in [VAC+03]. Here they outline the architecture for an intrusion tolerant, dynamic content Internet server as depicted in figure 2.3. They reuse the entities described above and in [VAC+03].

As we can see in figure 2.3, Saidane et al uses the basic architecture as that of Valdes et al, depicted in fig 2.3 They introduce a new system entity, an Adjudicator, who is in charge of access control and sanitation of the SQL queries received from the application servers. The adjudicator communicates with a fault tolerant database server. The adjudicator also ensures that the application servers submit the same SQL queries, linked to a HTTP request, in the same order, thus allowing consistency among the application servers. The

extensions they add to [VAC+03] allow update of the content while the system is online making it more flexible. By including functionality for dynamic content and databases, the system can support today's large online applications and add extra security. The dynamic web server uses the process depicted in figure 2.3 to process HTTP requests.

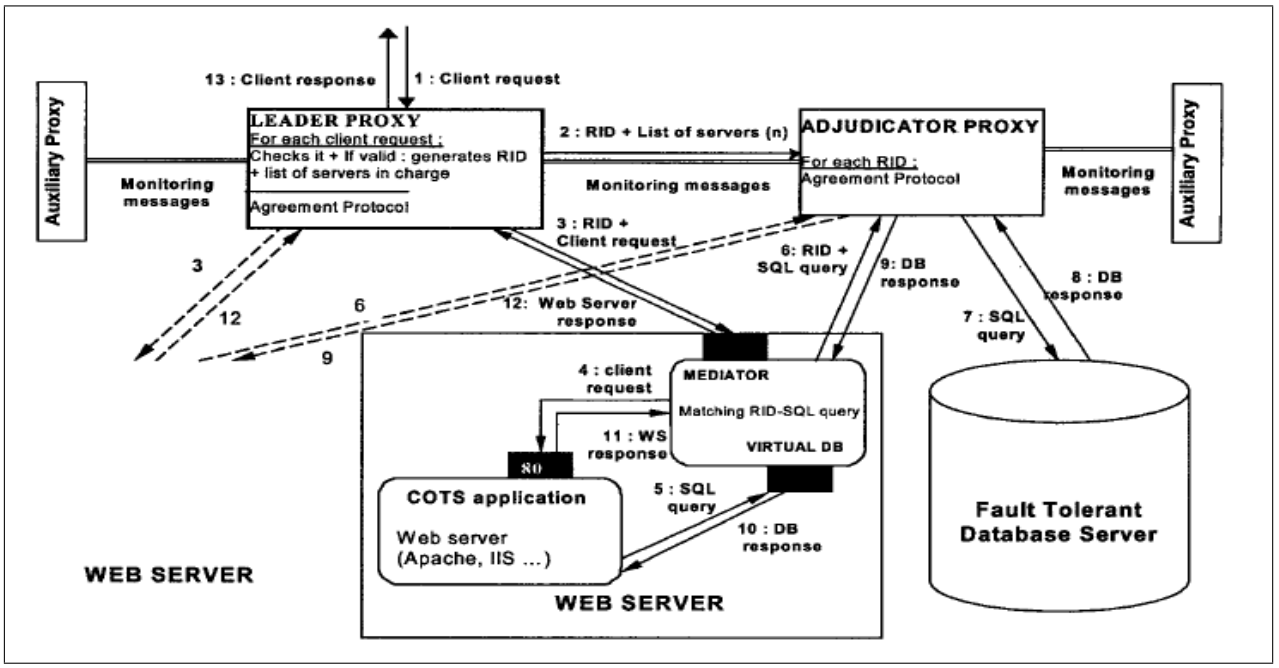

Figure 2.6: An overview of the dynamic web server mode of operation [SDN03]. The abbreviation RID is short for request identity.

Chapter 3

The system

In this chapter we will take you through the system setup and configuration. Since part of our assignment is to recreate and document the system constructed in [Nyr05], we will be quiet thorough about the details of the setup and configuration phase. This chapter is further divided into a system overview where we present you to the system and the setup part where we document the setup and configuration of the different system entities.

3.1 System Overview

The system depicted in figure 3.1 follows from the design and implementation in [Nyr05]. The system consists of 2 compartmentalized networks behind the proxy. One for handling HTTP requests and one, physically separate, for handling log traffic and incident reports. The proxy is reusing the MPITS system designed in [Bro05] and as in [Nyr05] the MPITS proxy, voter and logger functionalities is based in one physical entity. There are 3 Apache web servers running on the application servers. These servers are disk less servers, each with a dedicated file server for disk operations The communication between the application server and file server run on a dedicated NIC, as seen in figure 3.1.

Type	*eth*0	*eth*1	*eth*2
Application Server	10.X.0.1	10.0.0.X	-
File Server	10.X.0.2	10.0.1.X	-
MPITS Server	10.0.0.4	10.0.1.4	129.241.252.121

Table 3.1: The network configuration for each NIC in the system architecture. The X is substituted with the server unit number to obtain the full addresses.

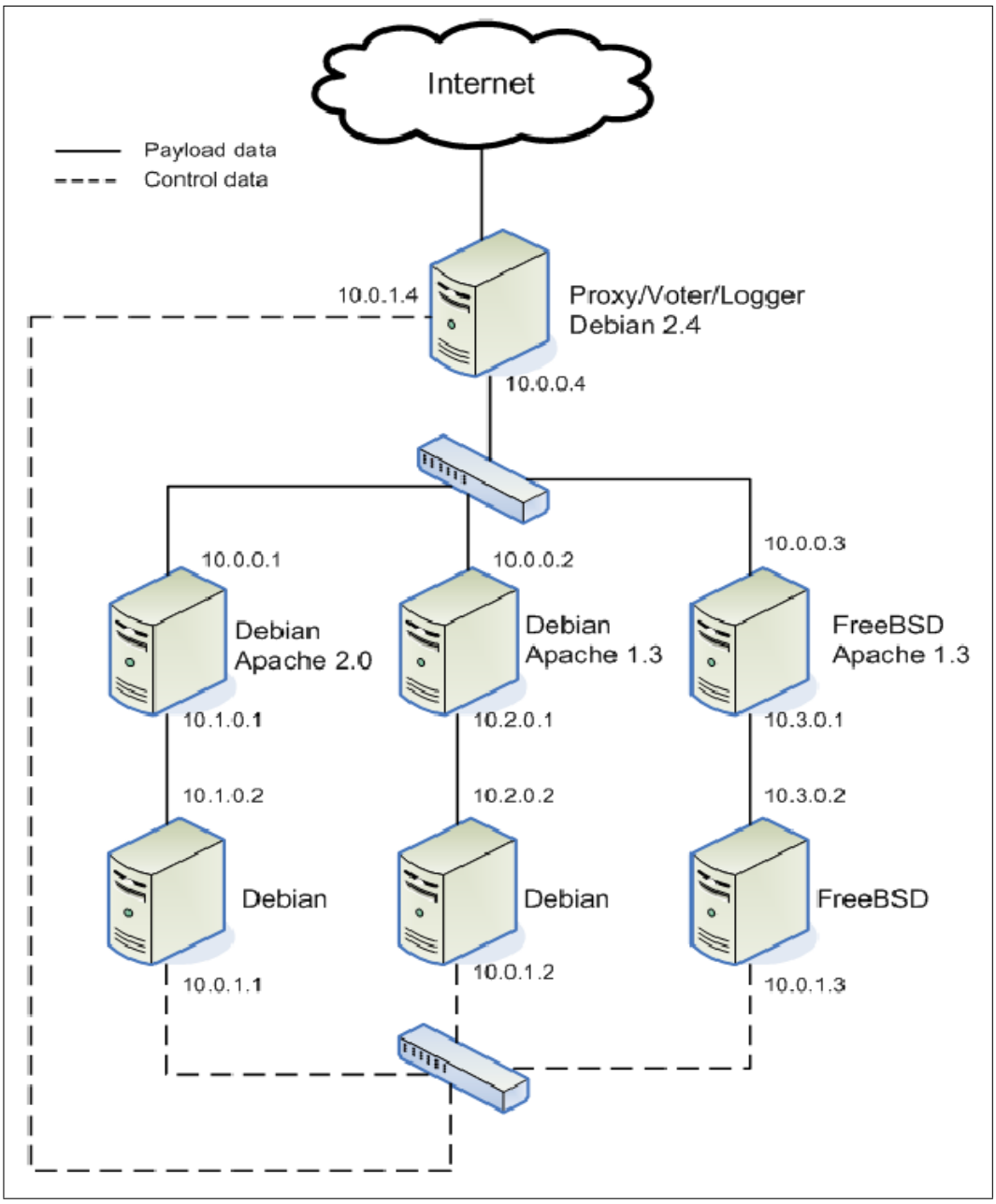

Figure 3.1: An overview of the system architecture

Type	*Manufacturer*	*Chipset*	*Memory*	*NIC*
File server	Unknown	AMD Athlon 1,4 GHz	1024 MB	2 X 3COM 100 Mbps
Application server	Unknown	AMD Athlon 1,2 GHz	1024 MB	2 X 3COM 100 Mbps
MPITS Server	Dell	Intel Pentium IV 2 GHZ	1536 MB	3 X 3COM 100 Mbps

Table 3.2: The hardware setup of the different server types

Applicationserver	*OperatingSystem*	*Webserver*
Server 1	Debian Sarge 3.2 (2.4 Kernel)	Apache 2.0
Server 2	Debian Sarge 3.2 (2.4 Kernel)	Apache 1.3
Server 3	Free BSD release 5.4	Apache 1.3

Table 3.3: The OS distributions and web server versions of the application servers

3.2 Hardware

In this setup of the dynamic honeypot system we use seven computers. Their hardware specification can be found in table 3.2. The usage of two separate NIC on each of the file- and application servers allows us to create two physically separate networks; One for control data and the other for web service traffic. The MPITS server uses the control network to alert the file servers of compromised application servers.

Due to the nature of this project it would have been desirable to run the system a diverse selection of hardware platforms. This would introduce diversity at the lowest possible level and guard the system against hardware specific exploits, but unfortunately due to limited equipment availability we can not explore this further. The Metasploit framework[9] contains several such HW exploits,

3.3 Operating systems, programs and protocols

The system is designed to run on diverse platforms, but due to problems with the installation of the application server images and time constraints we had to limit ourselves to the Linux and Unix systems in table 3.3. This is also the same Operating System (OS) used in [Nyr05]. We will give a thorough explanation of the problems we ran into during the setup and our solution in section 3.5. The application servers depend on Dynamic Host Configuration Protocol (DHCP) and Network File System (NFS) from the file servers to load the boot image. We will in this section give a little introduction to the OS and why we chose to use them. We will also introduce some of the components used in the system.

3.3.1 Free BSD

To introduce diversity at the kernel level we chose to use Free BSD. It is derived from BSD (Berkeley Software Distribution), the version of UNIX developed and distributed at the University of California, Berkeley. Free BSD is known to be extremely robust. There are numerous testimonials of active servers with very long uptimes. Free BSD has been the subject of a massive auditing project for several years, and is known for its security. This is the main reasons for why we wish to include Free BSD as a file- and application server OS.

3.3.2 Debian

Due to the complications mentioned in [Nyr05] about compiling a second kernel image fore network boot in Linux, we have chosen to adopt his tactic and base our work on the use of the Lessdisks program. The Lessdisks package is designed for use with Debian. Debian is a free GNU/Linux OS. Debian uses the Linux kernel, but most of the basic OS tools come from the GNU project; hence the name GNU/Linux. We use the same Debian distribution and kernel as in [Nyr05]. Debian uses a package system with precompiled programs in the Debian repository for installation. The package system in Debian takes care of dependencies between packages and automatically install the extra packages.

3.3.3 Lessdisks

Lessdisks is a ”diskless” terminal project designed for a standard Debian distribution. It contains a kernel bootable designed for diskless booting on a network Local Area Network (LAN). It was developed with the help of Freegeek [10] to continue the use of older hardware and reduce computer consumption. Lessdisks is a old technology, which appear to not be supported or documented at sourceforge anymore. But it is still available from the Debian package repository. This has been a complicating factor for us in this project. The last package update at the Debian repository for the Lessdisks packages was in may 2005. On the Lessdisks homepage there has been no update since April 2005, which was a bug fix, and the Lessdisks project seems abandoned. It is also worth noting that the newest release of Lessdisks is labeled 0.6, and there is no sign of any stable 1.X release ever made.

Lessdisks creates a new root file system at a designated place in the file servers directory. The new root file systems is administrated at the file server through the Lessdisks commands. When the application server boots, it gets an Internet Protocol (IP) address over DHCP from the file server Then the Lessdisks kernel image is transmitted over the network via NFS.

3.3.4 DHCP

The Dynamic Host Configuration Protocol (DHCP) provides a framework for assigning IP addresses and passing dynamical host configuration a TCP/IP network. It is based on the client server model where the DHCP server allocates network addresses and deliver configuration information to hosts. DHCP has been defined to be upwardly compatible with Bootstrap Protocol (BOOTP) [CG85] and both these protocols can coexist and function simultaneously in the same network. RFC 1534 [Dro93] defines how DHCP and BOOTP must be implemented to ensure they can co-exist in the same network and inter-operate. DHCP also supports interaction with PXE adding the capability of interaction with PXE booting clients. For more information about DHCP we refer to [Dor97].

3.3.5 UDHCP

The Debian kernel provided by the Lessdisks system uses the UDHCP (UDHCP) client package. The UDHCP protocol is a minimal implementation of the DHCP [Dor97], which is fully functional and primarily geared towards embedded systems. Due to its simplicity and small size it is ideal for a network bootable kernel. Since it it RFC compliant it can communicate with the DHCP daemon running on the file server.

3.3.6 PXE

The abbreviation PXE is short for Pre-boot Execution Environment, which allows booting a client off a network server, using the NICs ROM[11].. It was developed by Intel[12] to simplify the process of managing software changes on minimal computers and is incorporated in virtually all new NICs. One of the advantages of PXE is that once the client is set in PXE mode all subsequent configuration is done on the server side. It supports the following protocols and services TCP/IP, DHCP and Trivial File Transfer Protocol (TFTP).

3.3.7 TFTP

TFTP is a simple version of the File Transfer Protocol (FTP). TFTP uses the User Datagram Protocol (UDP). It is designed to be small and easy to implement and provides provides no security features. Its only purpose is to read and write files to or from a remote server and it acknowledges each package separately. It is often used, as in our case, by servers to boot diskless workstations, X-terminals, and routers. For more information about TFTP we refer to RFC 1350 [Sol92].

3.3.8 NFS

The NFS protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols. This portability is achieved through the use of Remote Procedure Calls (RPC) [SM88] primitives. This allows the server provide remote access privileges to a restricted set of clients and perform operating system-specific functions like read and write to the provided paths. For more specific information about NFS and the mounting procedures involved we refer to RFC 1094 [Inc89].

3.4 The file- and application server relationship

To understand the use of diskless application servers we need to explain the inner workings of such a system. This section is meant to give an overall picture of the boot process, the differences between Debian and Free BSD will be addressed in the next section. An illustration of the communication between application- and file server can be seen in figure 3.4.

At bootstrap, PXE, which is built into your NIC, will request a IPaddress over DHCP. The abbreviation PXE is short for Preboot Execution Environment, which allows booting Linux off a network server, using the NICs ROM[11]. Our file server has a DHCPdaemon and will assign an IP to the application server. If the DHCP server does not contain the pxeboot program the correct server can be provided by the *next-server* option and the file-name for the second-stage boot program. The PXE will then request and load a second-stage boot program, pxeboot over TFTP. TFTP is a simple version of FTP with no security functions. As the TFTP will function on a closed network, we do not feel the need to introduce more complex and security enhanced protocols by changing hardware. For more information about TFTP we refer to RFC1350[Sol92]. This program load modules and the kernel over TFTP and boots the kernel. A kernel is the central core of a computer operating system. We use the *vmlinuz* kernel, which is a compressed and bootable Linux kernel. Bootable means that it is capable of loading the operating system into memory so that the computer becomes usable and application programs can be run. The Kernel uses DHCP to acquire the network-configuration and then it loads the file system over NFS and proceeds with a normal boot from there.

3.5 Set up

In this section we will go through the setup of the system in a general manner and point out problems we have discovered, choices we have made and the solutions we have found. As said in section 1.5 there is little setup documentation available in [Nyr05].

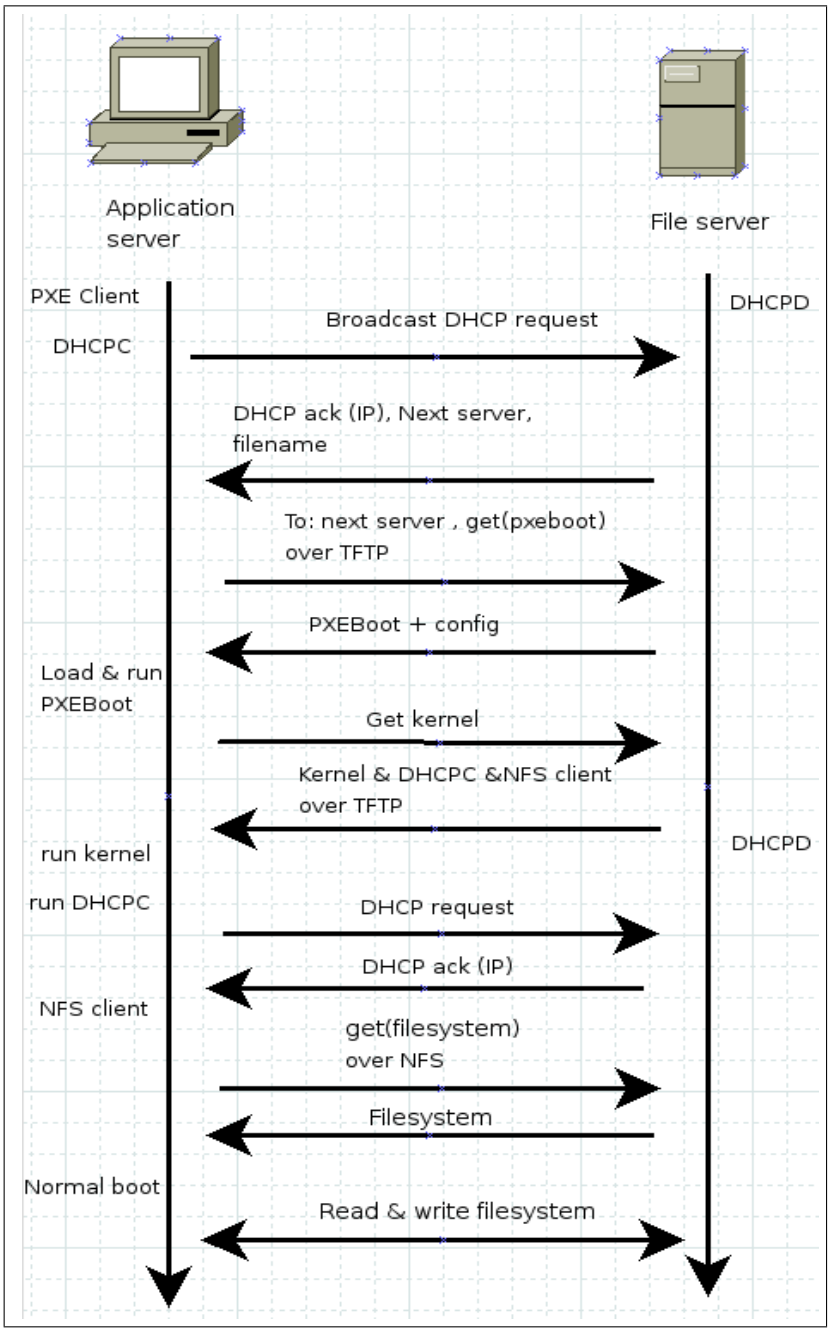

Figure 3.2: The communication between file and application server during boot

We have therefore included a complete and thorough walkthorugh of the setup in A. In this section it is important to note that we will be working as root (superuser) and that all paths are given as absolute paths from the root of the directory tree. If we use the *lessdisks-chroot* command and enter the application servers file system this will be stated explicitly. When it comes to packet managers, we have for Debian chosen to use *aptitude install* instead of *apt-get* as this deals with package dependencies in a more user friendly way. In Free BSD we will use the ports collection which we installed during system installation.

We will start from scratch by setting up the system backbone, the file servers, who provide the application servers disk-image. We installed Debian Sarge with a 2.4 Kernel on two of the file servers and installed Free BSD on the last one. We will first take you through the setup of the Debian-based servers, File server 1 and 2.

3.5.1 Debian File server

When we first started to set up the Debian machines we tried to use the Lessdisks repository for the latest version of Lessdisks, version 0.6.2d released April 2005, but it turned out that this version is corrupt. We tried all available versions of 0.6.2, the a,b,c and d but none seemed to function properly. So we used the latest stable release of the Lessdisks packages from the Debian package repository. So we edited *sources.list* according to this.

The version from the Debian repository seems to be 0.5.3 which is also the newest stable package available on the Lessdisks homepage, released June 6 th, 2005. We find it worth mentioning that versions 0.6.2x are all released before this package. After this release all activity on this project seems to have ended. Their page has been removed from sourceforge.net and the project seems to be closed. There is no record in [Nyr05] of which version that was used for their setup. From the dates available on the Lessdisks homepage, we suspect that they used either the same release we used or an earlier release.

Under the installation of Lessdisks you will be asked several configuration questions. We have included our configuration in B.1.2. It is important to install the *syslinux*, DHCP3 and the option *thy*. The syslinux package contains the bootloader used by PXE and some configuration files. Since we wish to make a diskless server the option *kernel-image-netbootable* must also be selected. For a more indepht explanation of this we refer you to appendix A.

We have used PXE to load the kernel over the network instead of the Etherboot program used in [Nyr05]. We explain our reasons for this change in section 5.2.1.

During a test of the boot environment we discovered that PXE on our 3COM card did not support Network File System (NFS), so we were forced to set up and use TFTP for the first part of the boot session. As seen in figure 3.4, the initial communication uses TFTP to transfer the PXEboot kernel, DHCPinformation and NFS. This is a somewhat more

complex solution, but due the hardware we have available it seems to be the only feasible solution. We wish to note that some newer NICs support the NFS PXE combination. We feel that our new setup does not remove us to far from the work of Nyre [Nyr05] when it comes to system functionality. Even though TFTP has no security, we feel that the use of TFTP at boot can be defended on a closed and controlled environment.

The application server uses a number of services as described in 3.4. We need to install, configure and initialize these services at the file server boot. We need DHCP to provide all the network-configuration needed to perform a network boot with PXE. The TFTP service must be configured and the export and access rights must be defined for NFS. Please read appendix A for the details of this setup.

After setting up the file server's services, we tested the application server to see if the system was working. When booting the application server we got an somewhat cryptic error message about a *spurious 8259A interrupt: IRQ7*. To rule out the possibility of a hardware error we tested three of our NIC's and found that they all responded with the same error message[1]. After spending a lot of time on debugging and googeling this error message we found a forum post stating that there is an driver bug for some versions of 3COM NICs in combination with the PXEclient and the 2.4 and 2.6 kernel. All 3COM NICs are of the 3COM 905CX series and seem to be affected with this bug. Following the thread on the forum we found a link to fix that was originally proposed by Peter Rundle [13]. It is a quite simple fix, which functions in the following manner: In the network configuration delivered to the PXEclient on the booting NIC we use an if/else construct which states that if the booting NIC has a on-board PXEclient it should request the pxelinux.0 bootloader instead of the ordinary bootable *vmlinuz* kernel. It seems that the 3COM PXEdriver does not realize that it needs more configuration data from the file server before loading the kernel so it must be directed to the bootloader. The bootloader will later, during the boot process, load the vmlinuz kernel as stated in 3.4.

When the Lessdisks kernel started to load it needed to regain its IP address. But when the Lessdisks kernel loads it stops when it should ask the DHCP daemon on the file server for an IPaddress. Due to the manner which the Lessdisks kernel stopped, we realized that it had to be an error in the script files used by Lessdisks to create the kernel. After reading the collection of bash script files at */etc/Lessdisks/mkinitrd/install_scripts/* we found a variable that should inform the kernel about using a DHCP client, in Lessdisks case UDHCP, to regain its IPaddress but this was *NULL*. We did not how ever succeed in finding the erroneous function which should set the variable in the scripts. But as we consider debugging Lessdisks as outside the scope of our project. We feel content with identifying the cause of this problem and chose to set the variable manually to DHCP and recompile the kernel. We realize that this is not the most elegant solution but due to time constraints we chose not to pursue a more elegant solution. For the specifics involving this error and recompiling the Lessdisks kernel we refer to A.

[1] We actually found one partially defect NIC while doing this test which we exchanged for an Intel card

3.5.2 Fileserver Free BSD

Entering the Free BSD world has proven to be a quite different experience then the Debian/Lessdisks combination. The general background and dependencies of a diskless operation is thoroughly documented in the Free BSD handbook[14] and the man page Diskless(8), even though the building of a diskless kernel is not.

Free BSD has a different approach to packagemanagmet then Debian Free BSD uses a Package Collection called Ports. Each "port" listed here contains any patches necessary to make the original application source code compile and run on Free BSD. Instead of acquiring precompiled programs we have to compile them ourselves. Normally, you can download the port, unpack it and run *make* and *make install* at installation time, but due to some connectivity problems because of an erroneous NIC we installed the entire ports hierarchy to disk.[2].

We need to configure the same basic services as on the other file servers. First we need Internet access so we temporary configure rc.conf with NIC names, IPaddresses and other network information. Then we enable DHCP, TFTP and NFS. For details on this setup we refer to Appendix A.3. The PXE booting, which caused difficulties for us under Debian, worked flawlessly under Free BSD.

It is important to know that Free BSD is a UNIX system and therefore there are several differences from a Linux environment. For the Free BSD setup we will, as in [Nyr05], not use any prebuilt kernel or helping software like Lessdisks. We wish to make a new root filesystem on a specific directory on the file server and mount this directory as the filesystem for the diskless application server. The *make world* command creates a whole filesystem inside a folder. This is a basic filesystem and does not include the ports collection or any configurations from the mother system. After creating the file system, we needed a kernel, supporting the diskless features we needed on the application server. Building the kernel turned out to be a very time consuming and error prone process as the diskless kernel feature is not well documented. To build a kernel you make a configuration file with the following:

- Machine architecture
- Kernel identity
- Options
- Device information

The machine architecture refers to which processor the kernel will be running on and allows us to enable hardware specific options. We needed to load modules supporting our hardware and the options. The kernel identity is the identification of the kernel and the value is printed when you boot the kernel. The options contain general configuration

[2]This has proved quite useful and time-saving due to program dependencies and we therefor recommend that you do this

information about kernel abilities like networking, filesystem, support and scheduling. The device information contains information about buses, controllers and hardware devices installed on the computer motherboard. For enabling diskless operation there are many options that must be loaded and all hard drive controllers and disk references (devices) must be removed. A complete *make buildkernel* and *make installkernel* takes approximately 2,5 hours on the available hardware. We have included a thorough guide on building a kernel for our hardware in appendix A in section A.3. Chapter 8 of the Free BSD handbook discusses general kernel configuration and building[15]. We have compiled a working kernel, but we have not made an optimized kernel due to the time constraints on this project. We can not rule out the possibility that our kernel contains security weaknesses that makes it vulnerable to exploits. Given time we would have wished to review our kernel and checked it against known module weaknesses.

3.5.3 Application servers

In this subsection we will deal with the setup of a application server in a generic manner. For system specifics we refer to appendix A section A.2 for the Debian set up and section A.4 for the Free BSD setup. First we must enable network booting in the Basic Input/Output System (BIOS) and set the NIC to boot over PXE. Then we must configure the NICs on each application server according to figure 3.1. After the basic configuration we must install the web servers, set listening ports and define the static web content on each application server.

3.5.4 MPITS

The setup of the MPITS is quite easy, we need to configure the NIC, compile the C code and configure the MPITS config file. The interfaces file with the NIC configuration is included in appendix B section B.5.1.

To get a more functional understanding of the inner workings of the MPITS, we have reviewed the source code and wish to explain some of the functions in detail. When the MPITS receives a response from all of the application servers it represents each response as a char-array and runs the equality algorithm implemented in the files equalityalg.c and equalityalg.h. The algorithm works by counting the number of equal characters in the streams as explained in section 2.2.5. Numbers are detected and a given percentual depreciation is allowed to compensate for any differences in the web servers implementation of HTTP. Responses judged different are kept in structs called equality classes. If the response is equal, the first is kept in an equality class and the second one is dropped. After processing all responses the MPITS performs a vote, and acts according to figure 2.2.5. The MPITS configuration file contains a number of settings which is important to comment in this section. The MPITS has three security policies which can be applied to the voter:

1. **All equal** all responses from the application servers must belong to the same equality class.
2. **Majority** more that 50 % of the responses must belong to the same equality class.
3. **Plaurality**, a relative majority of the responses belong to the same equality class.

The security policy is selected in the MPITS configuration file. The file also contains settings for the threshold limit 2.2.5, defining the upper limit of deviation for both numeric and char character representation in two PDUs. The MPITS will, as a security precaution, drop the root privileges after setting opp communication sockets with the application an file servers, and we must therefore specify the user name of an existing user to run as after the privileges are dropped.

3.6 SAMHAIN

The set up of SAMHAIN is independent of the underlying OS, so the installation and setup of SAMHAIN will be described in a generic manner. SAMHAIN can be downloaded from [16] and the documentation can be found at [17]. We will use the same setup and configuration as [Nyr05] to get a system that is comparable. This is important since part of our project assignment is to test and assess the functionality described in [Nyr05]. SAMHAIN contains a *make* file and the installation SAMHAINs client server functionality, but according to the samhainrc configuration file included in [Nyr05], all log traffic is emailed to root@localhost and this speaks against this. We feel that the log traffic should be kept at a different location and has therefore installed Yule at the physically same box as MPITS, but is is using the control data network, depicted in figure 3.1, for communication. Yule is the log server within the samhain file integrity monitoring system and receives all log data. It is important to note that the client and server are distinct applications, and must be built separately. Yule is a non-forking server. Instead of forking a new process for each incoming logging request, it multiplexes connections internally [8]. Apart from samhain client reports, Yule can also collect syslog reports by listening on port 514/UDP, but we have not chosen to install this option. Yule will as default store log reports in /var/log/yule. The client server communication is by default encrypted using the Rijndael 192 bits Advanced Encryption Standard (AES) algorithm but we turned off this option in our setup due to timeconstraints. It is important to set the log server address option in the samhainrc file to inform the client of the location of the server.

3.7 Problems and differences

We have had several problems during our recreation of Nyres [Nyr05] system. The Lessdisks program, running on the application servers, did not function until we altered

the source code of the program. Nyre does not comment on which version of Lessdisks they used or how they made it work and thus we can not be sure that we have created a similar system. We feel that our introduction of PXE has made the boot sequence more secure, as it does not rely on third party software and now is compliant with today's industry standard. The 3COM driver causing the *spurious 8259A interrupt* have also diverted us from the setup of Nyre. We therefore document that we have recreated the functionality of [Nyr05], but we also stress that we are probably not using the same services.

3.7.1 Data obfuscation Function

Nyre included a data obfuscation function which should switch the content of a specified directory of valid data with another of bogus data. [Nyr05] contains a bash script called *docroot.convert*, two files written i C, *server.c* and *server.h*, and a Binary file named server. As there is no documentation of how these files work we had to try and test them. First we copied the script to the /etc directory. In this script we can define the valid and the bogus directories. The script works by defining *symlinks* to the web servers directory from the valid directory, and when the script is called it exchanges the *symlink* with a link to the bogus data directory.

We think the data obfuscation function will be activated when the voter detects an anomaly in the responses from the application servers. Then the MPITS should send a message on the control network and alert the file server which should run the *docroot.convert* script and exchange the valid data with the bogus data. As we have no way of knowing the inner workings of the binary file we ran the binary file to see what it did. If can either be a compiled version of the C source files or it can be a supporting binary for the source code. The binary starts and then prints out: *success* and *could not look up user name* before returning the command promt. So we deduce that the binary is a compiled version of the C files. Inside the *server.c* we discover a variable called user name initiated to "pats". We change this to "jantore" and compile the source code and run it. Upon execution the terminal hangs and we can not get any responce from the file server. This also causes the NFS exports to the application server to time out. We used *grep* to see if the username variable is initiated somewhere else, with no result. We then created a user "pats" and compiled the source again, with the same result. We also tried with the "root" user with no luck. We opened an other terminal and ran *netstat*, we discovered that the code opens port 2400. But apparently the code monopolizes either the *inet daemon* or some other network related process on the file server and thus no other processes can use the network stack. We also tried to find out where the source code calls the *docroot.convert* script with both manual and automatic (grep) review of the source code but we never found any reference to the script. Due to these findings and the time constraints on our project, we conclude that the data obfuscation function do not function correctly on our setup. As for the reason to this we can only speculate as we do not know how Nyre designed this:

1. We observe that there is no link or system call between the bash script and the C code. This might indicate that we are missing some of the source code that nyre wrote, but as the code compiles without errors we find this less likely.
2. Our system setup could differ so much from that of Nyre, and thus creating an error in the communication between the source code and the OS.
3. Some other environmental or kernel conflict unknown to us might cause the function to fail.

We feel that the timeout of NFS and the following disruption of application server service is a serious flaw in our system and that this function should be reviewed or redesigned and tested thoroughly. This will be further discussed in section 5.5.4.

Chapter 4

Testing

We have in the previous chapter finished the setup of our system. Ideally, testing should have been preformed during the development of this system or on the actual system created in [Nyr05]. But as this system no longer is available to us, we have redeployed the system according to the description in [Nyr05]. As documented in 3.7, we have been forced to use several solutions seemingly not used in [Nyr05] to make the system work and therefore our system differs on several accounts. Even though the system differs, we feel that we have preserved the functionality and essence of the system and therefore can provide valuable test data to evaluate the system. In this chapter we will therefore conduct testing on the system we have deployed. We wish to find and identify flaws in the system and to come up with possible improvements.

4.1 Overview, methodology and test technique

There are many different test strategies we could deploy in this project. How to test and on what level we should test is a problem developers are faced with every day. A developer could use the test-driven development method where you first write a unit test, then the actual code, and then verify that the test runs to completion. On a higher level you can, according to to [VV02], test program functionality by the use of test sets. Test sets can be classified into three disjoint sets:

- **Coverage-based testing:** In coverage-based testing, testing requirements are specified in terms of the coverage of the product (program, requirements document, ect.) to be tested.
- **Fault-based testing:** Fault-based techniques focus on detecting faults. The fault detection ability of the test set then determines its adequacy.
- **Error-based testing:** Error-based techniques focus on typical error-prone points, based on knowledge of the typical errors that people make.

To problem with any test technique is to decide when to stop testing and what makes a test adequate.

Given time, we would have used the test strategy described in "The practice of System and network administration" by Limoncelli and Hogan[LH05], where they recommend the use of unit tests and automatic test scripts to verify that the basic services and functionality is provided. Even though we feel that such tests would be of great value due to the fact that they can be easily reproduced and rerun, the time constraints in this project do not allow us to use such a strategy.

On a higher level, both structural and functional testing is required to make sure that the system acts according to the specification. Structural analysis-based test sets tend to uncover errors that occur during the implementation of the system. Functional analysis-based test sets tend to uncover errors that occur in the requirements and design specifications. For additional information on testing we refer to [Per95]. Even though this book is ten years old we feel that it explains the principles behind testing very well. It is also the only book we have found that deals with security testing techniques, which we feel can be valuable to our project. As there is no specification available to us, only a loose list of requirements used by [Nyr05].We will therefore limit ourselves to testing the functionality described in [Nyr05]. We will also address the security functionality of the system.

4.2 Blackbox vs whitebox testing

Whitebox testing, also known as glass box, structural, clear box and open box testing is a software testing technique where explicit knowledge of the internal workings of the item being tested is used to select the test data. Unlike black box testing, white box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable.[VV02]. Most of our testing will be black box testing, but we will do a code review of the MPITS source code and a whitebox test of the basic web service.

4.3 Tests

We will perform several test to test important system functionality. We will describe the tests and document our results.

4.3.1 Test of basic web service

Our first test the voting mechanism of the MPITS to ensure that the basic functionality of our system is provided. When we send a request the system should respond with the correct web page within a reasonable amount of time. The first time the system responds correctly to our request with the correct web page. But when we press the refresh button in our browser, we get the system error page. If we press the refresh button once more we again have the correct page. The same sequence will repeat it self over and over again. When looking at the traffic between the MPITS and the application servers with Ethereal, we find the following: The MPITS establishes a TCP connection to each application server in sequence, queries for the web page and releases the connection before repeating the procedure for the next application server. We discovered that the problem lies with the application server running Apache 2. Apache 2 will respond with a HTTP 304 not modified message when refreshing a static web page. The 304 message is used when a resource is unmodified since the last request. Apache 1.3 on the other hand will not give the same message and therefore the MPITS voting mechanisms will draw an erroneous conclusion. In the w3 HTTP specification RFC 2616 [FBL99] we find the following concerning 304 message:

> If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code. The 304 response MUST NOT contain a message-body, and thus is always terminated by the first empty line after the header fields.

It seems that Apache 2 responds in the correct way, and that Apache 1.3 resend the status message 200 OK with a Hypertext Markup Language (HTML) payload and thus ignoring the RFC. This is not a system flaw, but rather something to be dealt with during the deployment of this system in a production environment. It is probably possible to change either Apache 1.3 or Apache 2 response to a refresh message, and thus avoiding the problem, but we will leave this for future work.

It is stated in [Nyr05] and [Bro05], that due to the fact that protocol specific meta-data may vary considerably, equality is not computed for this part of the PDU. We would think that the HTTP status code would be part of the protocol meta-data. We have included an HTTP header below, and in our opinion everything below *Connection: close* is meta data from the web server[1]. Also RFC 2616 [FBL99] refers to status codes as meta-information and this leads us to think that maybe the equality algorithm is not perfect in its current implementation.

```
GET / HTTP/1.1
Host: www.example.no
Connection: close
```

[1]The HTTP header originate from our system, but is used due to the amount of meta data included.

```
HTTP/1.1 200 OK
Date: Tue, 28 Nov 2006 11:14:36 GMT
Server: Apache/2.0.55 (Ubuntu) DAV/2 SVN/1.3.1
PHP/5.1.2 mod_ssl/2.0.55 OpenSSL/0.9.8a
Content-Location: index.en.html
Vary: negotiate,accept,accept-language
TCN: choice
Last-Modified: Sun, 29 Oct 2006 21:18:58 GMT
ETag: "e806c-efe-b38ab480"
Accept-Ranges: bytes
Content-Length: 3838
X-XRDS-Location: http://example.com/xrds
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Language: en
```

4.3.2 Session support in MPITS

We were uncertain if MPITS supported HTTP sessions in some manner. We therefore chose to do a code review of the of the supplied MPITS code. We have read [Bro05] and preformed a code review of the supplied MPITS code. From this we can conclude that there is no session support implemented in MPITS which means that each web server must maintain their own sessions. The issue of session support in MPITS is discuced further in section 5.3.1.

4.3.3 Testing SAMHAINs file integrity check

We decided to first do a basic test of SAMHAIN's functionality. In section 3.6 we set up integrity check and monitoring of important directories, including the *www* directory of the application server located at */var/lib/lessdisks/var/www/*. As it is critical for the voting procedure that this file remains uncompromised, we set up SAMHAIN to the highest integrity level, where access to the file is detected. We have had some concerns about a potential window of opportunity, that an attacker could use to harm the system before the change was detected. We therefore configured SAMHAIN to run checks with five minutes intervals and thus we had five minutes to access and alter the file before the alterations were detected. During this interval all web requests to the application server responded with our altered page. Afterward we wished to test SAMHAINs response time when running as a daemon, this spawns a process which continuous checks the files. It is possible to specify a check interval between every whole run.

We wished to test SAMHAINs response time when we accessed a file at an arbitrary

time. To do this test we first set enabled the daemon functionality and set the monitoring policy parameters and interval between scans to 60 sec in the config file located at /etc/samhainrc. We then created a new baseline for SAMHAINs database with the command:

```
samhain -t update -D --forever
```

The -D starts the daemon mode and enables it to run in the background and the parameter –forever ensures the continuous monitoring. When updating the SAMHAIN database, SHAMAIN sends a message to the root@localhost address. This message contains all the information about the files current state. The message contains information regarding the current state of the file; including access rights, group ownership, links, size, last access time and a checksum. These parameters are hashed into a file signature. We have included the message in section B.6.1.

Our test method here is simple, we access the file after waiting a arbitrary time, simulating an attacker using a directory traversal exploit. After accessing the file we got the email displayed in section B.6.2. SAMHAIN detected that the hash signature value had changed and included all integrity data in the email. We can see that *atime_old* differs from *atime_new* and due to this change the signature of first email does not match the signature included below.

When it comes to detection time, we note that the email was delivered at 16:36:11 and that we, according to the email accessed the file at 16:35:34. We repeated the test several times but we never got a response time above 60 sec. This might be because we accessed the file during an integrity check. We also looked into resource consumption of the SAMHAIN daemon. We used the *ps* command and found that when running a check the daemon uses 1,2 % of the CPU time. The CPU utilization increased to 14,3 % when adding the /usr directory which also contains the entire diskless directory. Due to the amount of SAMHAIN messages in our mail account after this trial, we confirmed that NFS accesses many files when loading a diskless application server. So some of the CPU utilization originates from sending the messages and calculating the changed hash signatures.

4.3.4 Testing the data obfuscation functionality

As the data obfuscation function do not work, we feel that there is little point in conducting testing beyond what is described in section 3.7.1.

4.3.5 Testing IPTables

We decided to test our our data control function, IPTables, by running *nmap* against MPITS to see if our IPTables script closed all ports. As we implement the data control

function described in section 2.2.3 through IPTables, it is crucial that this function properly. We got the following result:

```
Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2006-12-17 13:29 CET
DNS resolution of 1 IPs took 1.50s.
Initiating Connect() Scan against 129.141.252.120 [1679 ports] at 13:29
The Connect() Scan took 27.25s to scan 1679 total ports.
Host 129.141.252.120 appears to be up ... good.
All 1679 scanned ports on 129.141.252.120 are filtered

Nmap finished: 1 IP address (1 host up) scanned in 27.357 seconds
```

It seems *nmap*[18] do not detect the open port 80 for some reason, probably because it can not detect any of our web servers running behind the MPITS server. Our test verifies that all other ports are closed, as they should be. This result implies that our IPTables implementation are working as intended, blocking all traffic at closed ports.

Chapter 5

Improvements and further work

In this chapter we will explain what we perceive as system weaknesses and propose improvements to the system using the theory in chapter 2 to support our case. We regard this as the further work part of our assignment and has named it improvements and further work to fulfil both the areas of improvement stated in our assignment text and further work.

5.1 Overview

The rest of this chapter is structured in the following manner. First we explain our implemented improvements. Then we discuss our suggested improvements to MPITS, followed by suggested improvements for securing the system. These will realize the honey pot functionality we feel is missing from [Nyr05]. After dealing with security issues we discuss proposals for improving the system architecture, including tarpits. Even though tarpits clearly can be seen as a security feature we have chosen to discuss it here, as the feature is closely connected to the system architecture.

5.2 Implemented improvements

During our initial assessment of the system we identified some problem areas in the setup used in [Nyr05]. We therefore chose to implement what we judge to be improvements of the system during our setup.

5.2.1 The initial boot setup

In [Nyr05], it was decided to use the Etherboot project instead of the PXE. Etherboot is a software package for creating ROM images that can download code over an Ethernet network to be executed on an x86 computer. Many network adapters have a socket where a ROM chip can be installed. The code can be executed from ROM or it can be loaded, at boot, via a floppy disk or CD-ROM. On page 65 in [Nyr05] Nyre states that PXE would be a superior solution, but we quote:

> We could not get PXE to work properly, without being able to discover the cause of our problems, we shifted to using Etherboot.

We feel that the use of Etherboot is an inferior solution to PXE and have therefore chosen to implement PXE in our system. The use of PXE allows us to keep all the configuration on the server side and eases the complexity to the diskless setup. We also note that the Etherboot project have started writing gPXE, which is the next generation (and a rewrite) of Etherboot to enhance PXE compatibility, and add new capabilities[19]. As you can read in section 3.5 the conversion to PXE was not without problems but these problems were mostly vendor specific. We feel that by removing third party software from our system we have implemented a more well defined system.

5.2.2 Data control

We have identified a major weakness in the system described in [Nyr05]. There is no system functionality for limiting communication in and out of the system, nor within. Nyre does not discuss the concept of data control, as we described in section 2.2.3, at all. Data control deals both with external connection attempts as well as defining legal internal communication between the system entities. Without data control it is possible that one application server may be compromised and be used as a launchpad for attacks on the other application servers. If an attacker manages to compromise two application servers and manipulate the httpd server page, the MPITS system will believe that the remaining application server is compromised and accept a false response from the two corrupted application servers as a valid answer. In order to stop such a scenario we must not allow any traffic between the application servers. IPTables can be used to filter out unwanted and possible malicious traffic from the other application servers.

In the same manner, we do not wish any communication between an application file server pair, other that legal DHCP, TFTP and NFS traffic. If one attacker compromises an application server we must try to protect the integrity of the control data network by not allowing a file server to be compromised. We recommend that each host runs IPTables as illustrated in figure 5.2.2. The Linux operating system supports stateful packet filtering using *IPTables*. This software allows the kernel to inspect IP packets as they are received, sent or forwarded and make choices about what to do with them. We wish to implement *IPTables* on the interfaces that connect the file servers to the application servers. The

	App. server		Fileserver		MPITS		
	10.0.0.X	10.X.0.1	10.X.0.2	10.0.1.X	External	10.0.1.4	10.0.0.4
TFTP		69/tcp	69/tcp				
DHCP sr			547/tcp				
DHCP sr			547/udp				
UDHCP cl		546/tcp					
UDHCP cl		546/udp					
NFS sr			2049/tcp				
NFS sr			923/tcp				
NFS sr			2049/udp				
NFS sr			923/udp				
NFS cl		2049/tcp					
NFS cl		923/tcp					
NFS cl		2049/udp					
NFS cl		923/udp					
HTTP	80/tcp				80/tcp		80/tcp
LOGG				2400/tcp		2400/tcp	

Table 5.1: Table of open ports on each network interface. These are the only open ports on the different system types. The blocking is done by an IPTables script. Cl refer to client and sr to server.

application servers should run IPTables on the interface towards the hub to filter our traffic from the other application servers. The MPITS should also run *IPTables* on the internal interface to perform the function of data control as described in 2.2.3. This is critical to prevent any possible attacker from launching attacks on other systems from one application server.

Below we display one such IPTables script for the MPITS. First we declear the IPTables full path, to make sure that the $path is not forged. This is a common security measure Then we only allow traffic on the ports of the addresses seen in table 5.2.2 and deny all other traffic.

```
\#!/bin/sh
IPTABLES=/sbin/iptables
\${IPTABLES} -P INPUT ACCEPT
\${IPTABLES} -F INPUT
\${IPTABLES} -I INPUT -m state --state NEW -i eth0,eth1 -p tcp --dport 80 -j ACCEPT
\${IPTABLES} -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
\${IPTABLES} -I INPUT -m state --state NEW -i eth2 -p smtp --dport 2400 -j ACCEPT
\${IPTABLES} -I INPUT -i lo -j ACCEPT
\${IPTABLES} -P INPUT DENY
```

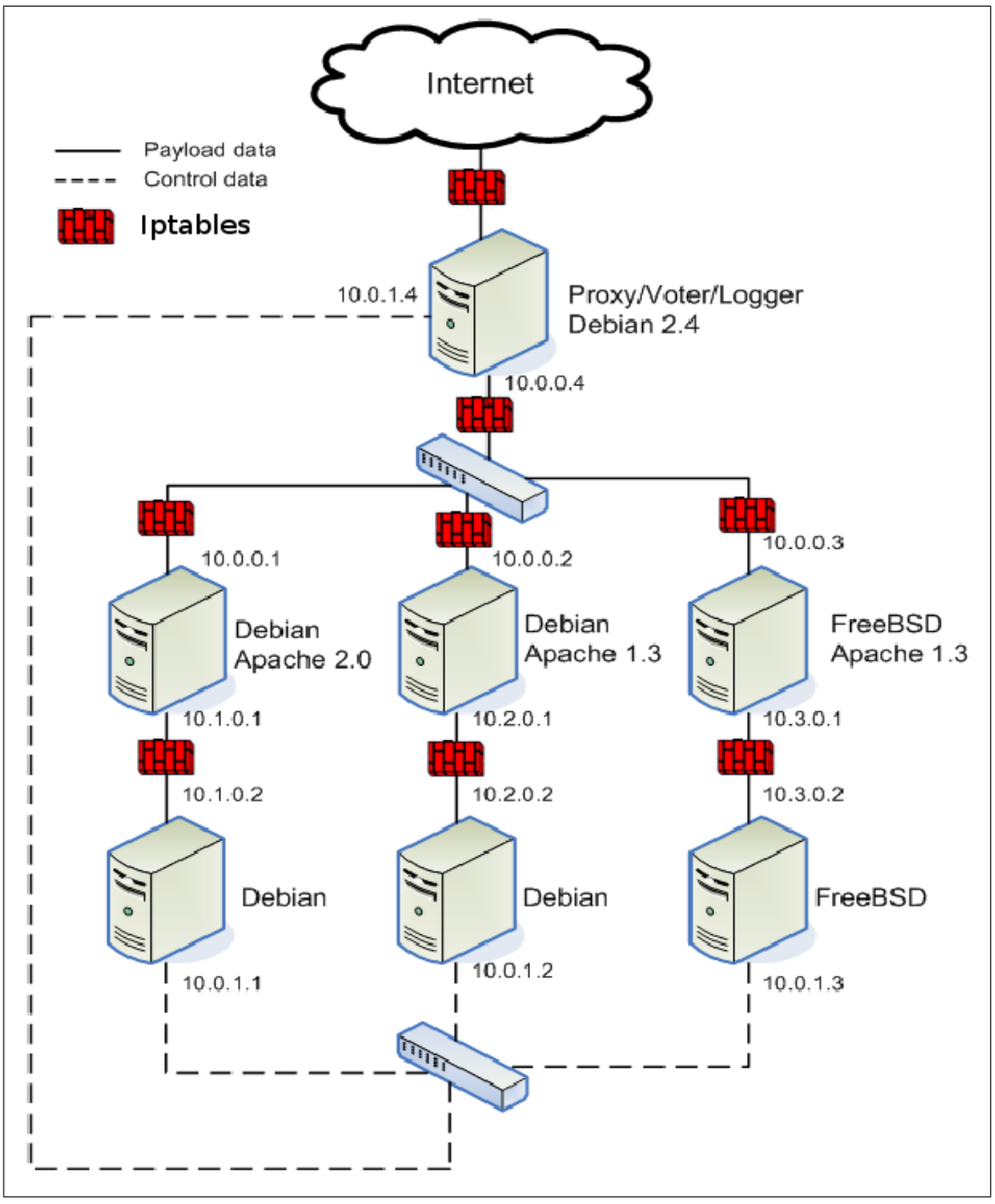

Figure 5.1: An overview of the system architecture with IPTables. IPTables is running on the interface above each host and on both payload data interfaces on the MPITS

5.2.3 SAMHAIN Logging server

We have implemented the SAMHAIN client server setup and thus realized the architecture as depicted in figure 3.1. We refer to this as an improvement as it is unclear whether [Nyr05] actually implemented this feature. By placing the logging server on a separate location we create a safe environment to keep and monitor the received log messages. The ideal placement for such a server would be on a dedicated log server, but following the original architecture of [Nyr05] we installed it on the MPITS server.

5.3 Suggested improvements to MPITS

We will in this section discuss the problems we have identified on the MPITS component and explain the reasoning behind our improvement proposals.

5.3.1 Sessions

We discovered during the code review of the MPITS code that MPITS only supports HTTP and has no session support. For a web service like our system this is no flaw, as all our web servers uses HTTP. We feel that the MPITS system does not need session support for normal use. The sessions can be transparent to the MPITS as long as all web servers maintain the state on each connection. The MPITS will just distribute requests to all our application servers, and thus each uncompromised and error-free application server will maintain the same state. The MPITS will contact all web servers, perform an evaluation vote on their response and forward one response to the client. All the MPITS should do is analyze the responses from the application servers and therefore the MPITS does not need to know the connection state of a client on an application level. This assumption should also apply for dynamic web content using databases, as all application servers should supply the same semantic response to the MPITS over HTTP. This should of course be tested, as we have seen, in section 4.3.1, that there are implementation differences in protocol semantics.

The problem arises when we wish to secure the web session by Secure Sockets Layer (SSL) or Transport Layer Security (TLS)[20]. Both SSL and TLS provides confidentiality and integrity of data while it is being transmitted through the Internet, but we will use the abbreviation TLS in this discussion. They use certificate schemes to encrypt data between two hosts in the network. The server side host must be the MPITS server in our case since we cannot encrypt the packets on the application server. This is due to the fact that encryption on the application server would yield three different responses from the application server and thus a tie in the voting on on the MPITS. Therefore the voting must take place before the encryption. We propose to let the MPITS include this functionality. This means that the encryption, TLS protocol handshake and key management must take place on the MPITS entity, and all session state must be kept

here. We realize that this will remove us somewhat from the idea of a Minimal Proxy for Intrusion Tolerant Systems (MPITS), as it will add complexity to the proxy. But it is our assessment that for the MPITS system to have any real value, it must support TLS over HTTP. Today TLS functionality is considered to be a basic security feature, and thus a "must have" for any web service. It is not an option to use the same private key on all three application servers as these are considered more insecure than the MPITS, due the minimal set of installed software on the MPITS entity. SSL or TLS functionality can be implemented on the MPITS e.g by using OpenSSL [20].

5.3.2 The control network should use broadcast

The use of broadcast in the back-end network could bring several benefits. In the control network, we could alert all other file servers that an attack is pending. The "I'm under attack" message should also include information about which file suffered an integrity loss and thereby allowing the other file servers to quickly run checks on these files to shorten the attacker's time window. It is also a good idea to provide extra monitoring of the files that was altered by the exploit to detect if any of the other application servers are vulnerable to the same exploit. The alert message should also trigger a script which could raise the monitoring level of these files to the highest level. Since monitoring levels are specified in a text file we think it will be a small task to write a script which listen for broadcast messages containing file paths on a specific port and raise the alert level of these files accordingly. Even though we use different *NIX OS, we feel that there are enough similarities between the systems to justify such a security precaution.

5.3.3 MPITS does not sanitize input or output

The MPITS server should contain functionality for sanitizing all meta-data for both incoming and outgoing requests. There are three reasons that justify sanitation of in- and output.

1. Any information about a system can help an attacker fingerprint the system and deduce possible vulnerabilities from the information. To limit the information about the system the MPITS should wash away any information about web server and OS before sending a response back to the client. An attacker could learn which web servers are running on the system by inspection of the HTTP response. In our current setup, the web server information in the HTTPresponse will change if an attacker presses the refresh button in her browser because, as we have stated in section 4.3.1, the server running Apache 2 will not be part of the same equivalence class as the other web servers as it will send a different response back to the client. If an attacker discovers that there are different web servers responding to a request and a refresh of the same page, then an educated attacker can deduce quite a lot about our system. To stop this information leak we should of course make sure

that all web servers respond to a refresh in the same manner and strip the HTTP response of any platform and server specific information.

2. If there is no input validation in the MPITS, malicious code can pass through to the application servers and potentially compromise them. We therefore need to validate all HTTP request before passing them on to server. The MPITS or some other system behind should look for anomalous HTTP requests to try to detect malicious payloads and prevent these from reaching the web servers. This detection mechanism could actually be used as a indication that that an attack is imminent. We recommend some sort of white list functionality to define which HTTP requests are valid, as this will be easier to implement than defining all sorts of unwanted input.

3. Given that an attacker has compromised an application server without detection and wish to leak information to the outside. She then can use the HTTP meta-data as a covert channel to send out information to the surroundings. We se this as another reason why the MPITS should replace all meta data with generic non-descriptive data before sending out a response.

5.4 Suggested improvements for securing the system

In this section we present proposals to secure the system. We focus on realization of the honey pot functionality we feel is missing from the system described in [Nyr05].

5.4.1 Further improvements of the SAMHAIN log server

We have, as described in section 5.2.3, installed the client server setup of SAMHAIN. We observe that there is no link between the Yule log server and the MPITS in [Nyr05]. We feel that there should be some functionality linking the two separate detection mechanisms. Alerting the voter that a application server is compromised, can allow the voter to discard a response from that application server. This link could be implemented through bash scripting. This alert could also trigger some separation mechanism that could segregate the attackers connection from legitimate connections. Such a mechanism should also log the attackers IP address to help identify the attacker and gather other information about the attacker. There are no such mechanisms implemented in MPITS today. Such mechanisms are discussed further in section 5.4.3.

5.4.2 Protection of log traffic

SAMHAIN clients can encrypt the log messages to yule, the SAMHAIN log server. By doing this we achieve an extra layer of security by adding the attributes of confidentiality and integrity to our log messages. The relationship between the SAMHAIN client and

server is explained in chapter 2 section 2.2.6. Even though the control data network is an enclosed LAN we feel that an extra security layer corresponds well with the principles of redundancy and diversity of security mechanisms.

5.4.3 Containment of compromised application servers

If one of the application servers becomes compromised, we need to contain this server from the other application servers. We also wish to separate the attackers connection from legitimate connections. To stop a potential attacker from listening in on legitimate traffic, the hub on the data network should be replaced with a layer 2 programmable switch with virtual LAN capabilities. By placing each application server on a virtual LAN, we achieve a redundant containment mechanism for containing each application server. An other solution is to allow the voter on the MPITS to alter the settings on IPtables to separate the attackers connection from other legitimate client. The separation of legal and illegal connections allow the system to continue functioning while deploying honey pot functionality against the attacker. Since one of the application servers has been removed from the voting procedure we feel that the security policy, described in section 3.5.4, should be set to the all equal to ensure correct replies to clients. We wish to point out that there is a flaw in the way Nyre thinks about the system security policy on the MPITS. Given that the system uses the stringent security policy, all equal, and one of the application servers becomes compromised, the MPITS will detect this through the voting procedure as there is no link between the SAMHAIN subsystem and the MPITS voter. We already discussed the need fore such a link in section 5.4.1. When the MPITS detects that the compromised application server's answer differs from the others under a all equal security policy, the MPITS will print an error message and shut down. And thus removing any shred of survivability from the system as the attacker can take down the entire system by compromising one application server. We therefore stress that the correct strategy for MPITS is to first use the majority security policy and when the voter detects an anomaly from one of the application servers, the policy should be changed to all equal as there no longer is an odd number of application servers. The compromised application server should, as stated above, be excluded from the voting procedure. The functionality for excluding compromised servers and changing security policies based on voting results, should be implemented in the MPITS voter. Only by utilizing the strategy outlined here can our system achieve survivability. The MPITS should not be able to use the all equal security policy when there is a odd number of uncompromised application servers running. The containment of compromised servers will also prevent the possibility of two compromised application servers forming a false majority.

5.4.4 Detection of changes in system state

As explained in section 2.2.2 there are several ways to detect an intrusion. We hope to detect attacks by detecting unwanted changes in the system state. We use SAMHAIN for this. SAMHAIN creates a hash of important files and storing these values in a database. To detect changes SAMHAIN recomputes these values and compare them to the database value. Since this is, by default, done in a periodic manner we feel that this leaves a window of opportunity for an attacker to tamper with the files before the system becomes aware of the intrusion and can deploy the countermeasures like data obfuscation. One attack on a similar system, Tripwire, is documented in [Acc99]. Even though the database is on a secure server, the file server, we fear what a competent attacker can manage to damage the application server within such a time frame using automated scripts. We therefore wish to point out the need for real time or close to real time change detection. This can, as we have shown in chapter 4 section 4.3.3, be achieved to a certain degree by running the SAMHAIN daemon continuously.

5.4.5 Increasing system surveillance/monitoring

[Nyr05] is titled *Increasing Survivability by Dynamic Deployment of Honeypots*, in our opinion the honeypot functionality is never deployed. We refer to the description and definition of a honey pot in section 2.2 and argue that there in Nyres system lies no value in allowing the system to be compromised. We need to monitor the attackers actions on a much closer level to learn anything from her actions. By monitoring changes in system state we learn nothing about which programs the attacker installs as they may not be installed in a monitored directory. We know nothing about which commands the attacker uses if he gains root access through some exploit. We therefor propose to install and run a Sebek [5] client on each application server and install the sebek logging server on the file server. Sebek logs all commands and keystrokes used by the attacker as we as the processes run on the monitored server. Sebek processes may be hidden from the attacker by the use of root kit techniques. Data captured by the Sebek client is sent to the Sebek server using a UDP covert channel. Sebek uses its own kernel-based implementation of the Raw Socket interface, and thus bypassing the OS network stack making it very difficult to detect the communication between server and client.

5.4.6 A dynamic Honey pot system

We have in this section come up with proposals to include the functionality we feel is missing the Nyres system. We have focused on the issues of data control and data capture, as these issues are closely related to the core principles behind a honeypot. To improve data capture we have proposed the following improvements:

1. Seperating the attacker connection and logging her address in the MPITS.

2. Using ofsite logging running the SAMHAIN log server on the MPITS.
3. Increasing the monitor granularity in application servers to logging keystrokes running processes and commands by using Sebek.

When it comes to data control we have implemented a strict flow regime through IPtables. We also, in section 5.3.3, discuss the need for input and output validation of the HTTP traffic. With these mechanisms in place we feel that we have full control of the data flows.

By including these functions we feel the system can justifiable be called a honeypot system. It is still lacking means to further identify an attacker, beyond her IP address, but we now have the means to learn from an attackers actions.

We have made these proposals with our current architecture in mind, but we wish to point out that by placing all log servers such as Yule Sebek on the Minimal Proxy for Intrusion Tolerant Systems (MPITS) the proxy can not be referred to as minimal anymore. We therefor propose a new entity to the system, a separate log servers on the control network to process all log traffic. By limiting the number of services running on MPITS we preserve the idea of a minimal proxy from [Bro05]. The MPITS should still run an IDS, Snort, to detect possible malicious code as a third detection mechanism. We discuss IDS further in section 5.5.3.

5.5 Proposals for improving the system architecture

We will in this section discuss the architecture used in [Nyr05] and present several different solutions and changes that we think will improve the system's value without removing us from the basic idea behind the system.

5.5.1 Number of entities

We feel that it is possible to cut down on the number of back-end file servers who serve the diskless servers. It is quite possible to have two or more application servers served by the same file server. This should of course be tested with respect to performance and capacity capabilities of the file server, but we feel that it provides a more scalable solution and leaves us with several interesting scenarios:

1. we can increase the total number of application servers to a larger odd number and alter the MPITS accordingly. This will, in today's solution, be limited by four things.
   (a) The number of available diskless application servers. Since part of the idea behind using diskless servers is to reuse old and outdated equipment this is not the most limiting factor. It is also quite possible to use thin terminals

for application servers, but this removes us a bit form the concept of using COTS.

(b) In its current design the MPITS queries in sequence each application server, stores the response and then performs a vote on the gathered answers. For a large number of application servers the query and vote delay will place an upper bound on the number of application servers. This can of course be fixed by using some sort of parallel thread function to query elements on the payload data network.

(c) A cost-benefit analysis to evaluate the gain of higher survivability against the cost of equipment, space, power, Heating, Ventilating, and Air Conditioning (HVAC) systems should be performed to find a suitable solution for any given deployment environment.

(d) The introduction of n application servers on each file server will in today's detection strategy lengthen the attackers window of opportunity w, by a worst case of $n*w$. We feel that this is a security risk to the entire system and needs careful consideration. We refer to our discussion of improving the detection functionality in 5.4.4.

2. We could decrease the number of back-end file servers. As seen on [21] it is possible to recompile Debian specific packages to run on Free BSD. By using the Free BSD kernel and libc, it is possible to recompile Debian packages to work on a Free BSD system, so we could change the Lessdisks package to run on Free BSD. By doing this, one file server can support all three application servers. As mentioned earlier this will increase both the load on the file server and the period between each integrity check of a specific file. We discuss another option in 5.5.5 where we eliminate the Lessdisks program.

3. Apart from the load and and system integrity checks objections we can not se any reason why we can not let each file server support several application servers each running a different OS, kernel or web server, and thus greatly increasing the system survivability by including more *NIX distributions.

5.5.2 A new architecture

We also feel that we should discuss the need for the diskless application servers and the back-end file servers. Can we reduce these to one entity and still keep the wanted functionality? The current architecture limits the use of this system in a production environment. Even though the system is based on reuse of outdated COTS components, we feel that the cost and complexity of running this architecture in a production environment will be prohibitive. We refer to the operating expenses such as space, power, HVAC systems as cost in this setting. But for companies all ready in possession of such a controlled environment and with stringent demands for security this solution

can still be economically justifiable. They may already be in possession of the facilities and the advantage of increased integrity, and availability through the use of (possibly outdated)COTS could make our solution economic viable. We will still discuss some alterations to our system that could reduce deployment cost. We could eliminate the diskless feature and the file servers by using the architecture described in section 2.3 and by Valdes et al in [VAC+03], but that involves the use of several tolerance proxies to monitor all system entities, so there is little to gain with respect to the number of COTS machines used. We therefore wish to propose a different solution. One of the file servers tasks is to run integrity checks on the system state to detect changes. In our system we use SAMHAIN[7]. The reference values of the system state is kept in a database which is queried during system integrity checks. The main reason for keeping this functionality of the application server is to store the database at a secure location. This prevents an attacker from tampering with the reference data as described in [Acc99]. Another way to maintain a completely safe storage solution for the file integrity database could be to keep the database at a read-only medium for instance a CD-ROM, thus eliminating the file servers completely. The risk by this solution is that an attacker might manage to unmount the CD-ROM and thus effectively stopping the integrity check. This can of course be prevented by monitoring the files */etc/fstab*, */etc/mtab** and */dev/cdrom*. One other possibility is to use SAMHAIN in stealth mode, meaning that it runs without any obvious trace of its presence on disk. If an intruder does not know that SAMHAIN is running, she might not make any attempt to subvert it. This technique uses kernel modules to hide samhain. In stealth mode the run-time executable will hold no printable strings and the configuration file is camouflaged as a postscript file with uncompressed image data, wherein the configuration data are hidden by steganography. There are similar techniques for hiding the database and log files by appending the data to an existing image file and xor the data with some constant to mask them as binary data[22]. [1]. By using the SAMHAIN stealth mode, we can eliminate the need for a file server as a safe storage medium for the file integrity database, and thus reduce the complex architecture of [Nyr05]. This leaves us with the MPITS server and back-end application server, which we feel is a much more realistic architecture and perhaps so cost-effective that it might be profitable to deploy instead of custom equipment. We will address the discussion of COTS versus custom equipment later in this chapter. But this change introduces some caveats.

1. The use of kernel modules to hide running processes is, with the Sony BMG's "DRM rootkit" incident fresh in mind, frowned on because this module will hide hide any process or file containing a specific name. SAMHAIN lets you choose the name during the installation with the option:

   ```
   --enable-install-name=NAME
   ```

   It is therefore **very** important to change the executable name to something that is

[1] Since most of this information can be found in the SAMHAIN man page we would not recommend installing the man pages in a production environment

really difficult to guess. And there even without the knowing the name, a suspicious attacker may use methods to detect the rootkit as seen in [23] .

2. The modules are kernel-specific, and they must be recompiled whenever the current kernel is recompiled, updated or replaced by another one. This even applies if the kernel version is identical. Failure to recompile might lead to a kernel panic.

5.5.3 Production environment with use of sticky honeypots / tarpits

A normal production environment will have little use for a normal honeypot. A honeypot system requires vigilance and tuning, and in a production environment one seldom have time and resources to perform such tuning. In such environments the primary objective is to stop the attacker from doing damage. To effectively stop the attacker we propose to use the tarpit technology of Labrea mentioned in section 2.2.1. If we can stop the attacker connection to the compromised application server by hanging the attacker's TCP/IP connection, this would unquestionably prove an asset for a production environment. This would mean implementing some changes in both the MPITS and the file server. We would have to change the MPITS server to exclude a compromised application server from the legitimate servers and allow the MPITS or the application server to tarpit the connection. To achieve this tar pit state, we allow a tarpitted port to accept any incoming TCP/IP connection. IPTables accepts the connection and then switches it to a zero-byte window. This forces the attacker's system to stop sending data. Any attempts by the attacker to close the connection are ignored, so the connection remains active and typically times out after 12 to 24 minutes. The question is who should have this functionality. It is obvious that the MPITS external interface could run this feature, to stop any worm or other automated connection attempt. We still need mechanisms to identify the attacker. Identification of an attacker can partially achieve with the use of Sebek as stated in section 5.4.5. But Sebek can only detect attackers who actually succeed in their attempt to compromise a host. To also detect intrusion attempt through malicious HTTP payloads, we propose that the system deploy an IDS like Snort[4]. The IDS should communicate with MPITS to alert the proxy of such payloads, enabling the proxy to prevent such payloads from reaching the web servers.

We realize that the use of IPtables as an active self defence mechanisms on the Internet is controversial and in a grey area of the law. We feel that it is acceptable as a hung TCP connection uses very little system resources and it is up to the connector to initiate these connections. System resources is a easily renewable resource and the loss of computing power is only temporary, as the tarpit does not do any permanent damage, it only hangs the TCP connection until it times out.

5.5.4 The data obfuscation function

As we explained in section 3.7.1, the data obfuscation function does not function correctly on our system. We therefore wish to point out the need fore further investigation of this function. Whether such a function is a valuable addition to our system depends on the functions ability to do the change undetected. This function should be subjected to analysis or possibly redesign to achieve the functionality described in [Nyr05]. Due to time constraints we have not looked into the security of using *symlinks* to exchange data directories but we think that mounting and unmounting directories might be a more appropriate solution in combination with NFS.

5.5.5 Removing third party software

We feel that the setup of [Nyr05] rely to much on third party software e.g Etherboot and Lessdisks. Such software pose a security risk as it is written by outstanders and we really do not know the inner workings of such software. We have already removed Etherboot, by recreating the system using PXE. We have already stated in section 3.7, that we had to make alterations to the source of Lessdisks to make it function with DHCP. We therefor wish to propose a new set up for the Debian file servers. In this solution we eliminate the Lessdisks program, instead we create a second partition on the file server and install the whole OS for the application server there. Then we mount the partition and export it using the PXE/NFS solution we described in section 3.4. After installing the OS we must reconfigure the kernel and load the diskless options, then we must recompile the Debian kernel to apply our new options.

5.5.6 The use of cots vs custom equipment

As stated in section 1.2, the system is based on the idea that redundancy and diversification in components can increase a the survivability of a system through the use of Commercial off-the-shelf (COTS) over special hardware. Specialized hardware equipment is a legacy from the telecom world and fulfil the "five nines" requirement, but each nine come at an increasing cost[2]. Even though such equipment can provide almost full availability and availability is one of the criterion of security, we feel that our system can provide much of the same service at a much lower cost. There are very few companies who have "five nines" availability demands, and therefor our system should be able to adequate availability with the extra security features introduced in our revised system. As stated in section 2.1.2, the focus of our assignment is not to measure the system's dependability. Other issues such as load balancing should also be considered. We only wish to point out the need for such an evaluation, including a cost-benefit analysis, as a part of our suggestions fore further work.

[2]The five nines refer to system up-time or availability as 99.999%, or an approximate downtime of 5 minutes per year

Chapter 6

Discussion and Conclusion

In this chapter we will do a brief comparison and evaluation of our system against that of Valdes et al described in section 2.3 and draw some conclusions based on our work.

6.1 System evaluation

If we compare the architecture of Nyre in figure 3.1 with that of Valdes et al in figure 2.3, we see that the basic architecture is quite similar. Both architectures use back-end application servers to provide content to a proxy, which distributes the data to the client. Each system uses a two tire detection mechanism to detect anomalous behavior in the system. Valdes et al uses a *content agreement protocol* and several IDS monitors to detect changes and anomalies in the data from the application servers. They also introduce the concept of tolerance proxies to monitor all system entities. Among these proxies a leader is selected through an election procedure[VAC+03]. The leader is responsible for filtering and sanitizing client requests and load balancing among the application servers. Nyre [Nyr05] and Broen [Bro05], on the other hand, rely on a much simpler detection mechanism, a voter and file system integrity check to detect change. Nyre continues the idea of simplicity in the proxy introduced by Broen in [Bro05]. Nyre's proposed function for data obfuscation could bring added value to the system given that the system contains confidential data. Both systems use the same amount of equipment to achieve the same basic web service, so we judge the operating expenses of the systems equal as both use COTS. As Valdes et al rely on complex voting and *content agreement protocols*, we feel that their setup requires more computer resources and is a more expensive option. All things considered, we feel that Valdes et al introduces to much complexity in their system. A simple system can be much easier understood, and thus considered more secure.

As neither of these systems support dynamic web content, we wish to to mention the continuance of the work in [VAC+03] by Saidane et al[SDN03] described in section 2.3.

The option of providing dynamic web content makes this system superior to the others. We still wish to point out that the architecture in [SDN03] has added more complexity to the already complex system of [VAC+03]. Saidane et al no longer use the principles of redundancy and diversification at all levels of the system architecture, as they have installed a single fault tolerant database as the back end system. As stated in section 5.3.1, our system should be able to support dynamic web content and databases without alterations, as all application servers will receive the same requests and supply the same replies. This allows us to extend our system architecture with one database on each application server, providing the options of redundancy and diversification at this level as well. This will not introduce more complexity to our system architecture as all web application complexity will be dealt with at web server and database level. The key issue for our system, is that all responses from the application servers should be semantically equal. We therefore feel that our system achieve the same functionality without the increasing the system complexity. We rely on simple mechanisms to provide the same service.

6.2 Conclusion

In this project we have focused on recreating the system described in [Nyr05]. We have documented our system setup and thus made it possible for others to experiment and verify our findings on an identical system. We have shown that we through simple mechanisms can create an intrusion tolerant system. We have identified areas of improvement for the system, and performed tests to support some of our findings. We have also made our own contribution to the system by implementing several improvements to the diskless boot procedure and to the honey pot functionality. By removing some of the third party software used in [Nyr05], we feel that the system now is easier to understand and to compare with other systems. We have installed a centralized log server, yule, to coordinate alerts and provide off-site storage for these log messages and IPTables for data control. By implementing these extended data control and data capture functionalities, we feel that we have realized some of the honeypot functionality missing from [Nyr05]. We have made many suggestions on how to further improve the system for deployment both in a scientific and a production environment. We feel that these suggestions should be implemented and tested to provide quantitative results for further work.

Bibliography

[Acc99] Phreak Accident. Playing hide and seek, unix style. *Phrack Magazine*, 4(43):14 –25, 1999.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11 – 33, 2004.

[Bro05] T. Broen. Innbruddstolerante systemer, en eksperimentell utprÃ¸vning og vurdering. Master's thesis, University of Oslo, Norwegian, May 2005.

[CG85] B. Croft and J. Gilmore. Bootstrap Protocol. Internet Engineering Task Force: RFC 951, September 1985.

[DeL06] L.L. DeLooze. Attack characterization and intrusion detection using an ensemble of self-organizing maps. *2006 IEEE Information Assurance Workshop (IEEE Cat. No. 06EX1287C)*, pages 108 – 15, 2006//.

[Den87] Dorothy E. Denning. Intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222 – 232, 1987.

[Dor97] R. Dorms. Dynamic Host Configuration Protocol. Internet Engineering Task Force: RFC 2131, March 1997.

[Dro93] R. Droms. Interoperation Between DHCP and BOOTP. Internet Engineering Task Force: RFC 1534 (Standards track), October 1993.

[EM99] Fisher D. A. Linger R. C. Lipson H. F. Longstaff T. Ellison, R. J. and N. R. Mead. Survivable network systems: An emerging discipline. Report by SEI Joint Program Office, 1999.

[FBL99] Gettys J. Mogul J. Frystyk H. Masinter L. Leach P. Fielding, R. and T. Berners-Lee. Hypertext Transfer Protocol: HTTP/1.1. Internet Engineering Task Force: RFC 2616, June 1999.

[Gla11] R.L. Glass. The software-research crisis. *IEEE Software*, 11(6):42 – 7, 1994/11/.

[Hel03] B. E. Helvik. *Dependable Computing Systems and Communication.* Tapir akademisk forlag, Trondheim, NTNU, 2003.

[Inc89] Sun Microsystems Inc. NFS: Network File System Protocol specification. Internet Engineering Task Force: RFC 1094, March 1989.

[JH02] Jaatun. M. J. and G. Hallingstad. Techniques for increasing survivability in nato cis. *Proceedings of the 1st European Survivability Workshop*, 2002.

[Lap95] J-C-. Laprie. Dependable computing and fault tolerance: concepts and terminology. In *Digest of Papers - FTCS (Fault-Tolerant Computing Symposium)*, pages 2–11, 1995.

[LH05] T. A. Limoncelli and C. Hogan. *The Practice of System and Network Administration.* Addison Wesley, New York, USA, 2005.

[Nyr05] å. N. Nyre. Increasing survivability by dynamic deployment of honeypots. Master's thesis, Norwegian University of Science and Technology, September 2005. Available on request from the Department of Telematics, NTNU.

[Per95] W Perry. *Effictive Methods for Software Testing.* Wiley-QED Publications by John Wiely and sons, Inc, New York, USA, 1995.

[SDN03] Ayda Saidane, Yves Deswarte, and Vincent Nicomette. An intrusion tolerant architecture for dynamic content internet servers. *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems*, pages 110 – 114, 2003.

[Shi00] R. Shirey. Internet Security Glossary. Internet Engineering Task Force: RFC 2828 (Informational), May 2000.

[SM88] Inc Sun Microsystems. RPC: Remote Procedure Call Protocol specification: Version 2. Internet Engineering Task Force: RFC 1057, June 1988.

[Sol92] K. Sollins. THE TFTP PROTOCOL (REVISION 2). Internet Engineering Task Force: RFC 1350 (Informational), July 1992.

[Spi03] L. Spitzner. The honeynet project: trapping the hackers. *IEEE Security and Privacy*, 1(2):15 – 23, 2003/03/.

[SSl06] I. Solheim and K. Stø len. Teknologiforskning - hva er det? Technical Report STF90 A06035, SINTEF IKT, SINTEF IKT, Norge, September 2006.

[Sto89] C. Stoll. *The Cuckoo's Egg.* Doubleday, New York, USA, 1989.

[VAC+03] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T.E. Uribe. Architecture for an adaptive intrusion-tolerant server. *Security Protocols. 10th International Workshop. Revised Papers (Lecture Notes in Comput. Sci. Vol.2845)*, pages 78 – 158, 2003//.

[VV02] H. Van Vliet. *Software Engineering, Principles and Practice*. Wiley, New York, USA, 2002.

[ZW05] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23 – 31, 1998/05/.

Chapter 7

Web resources

We have verified that all web resources exsist on the Internet today, December 20, 2006.

1. Merriam-Webster's thesaurus
   http://www.webster.com

2. Lessdisks homepage. Diskless tool for Debian
   http://www.lessdisks.net

3. Labrea homepage at sourceforge. Tarpit honeypot
   http://labrea.sourceforge.net

4. Snort Homepage. Intrusion detection tool
   http://www.snort.org

5. Sebek Homepage. Data capture tool.
   http://www.honeynet.org/tools/sebek

6. Nepenthes Homepage. Versatile honey pot tool for collecting malware.
   http://nepenthes.mwcollect.org/

7. SAMHAIN manual. On initialzing the baseline.
http://www.la-samhna.de/samhain/manual/installation-initialize.html

8. SAMHAIN Log server maual. On Yule, the log server
http://www.la-samhna.de/samhain/manual/yule.html

9. Metasploit framework homepage. Userfriendly exploit tool.
http://metasploit.com/svn/framework3/trunk/modules/exploits/

10. Freegeek homepage.
http://freegeek.org/

11. Syslinux homepage. Linux help page.
http://syslinux.zytor.com/pxe.php

12. PXE specifications.
http://www.pix.net/software/pxeboot/archive/pxespec.pdf

13. Lessdisk Forum. The Peter Rundle DHCP fix.
http://dev.lessdisks.net/projects/lessdisks/browser/tags/0.6.2c/doc/pxe/dhcpd.conf?rev=1873

14. Free BSD handbook. Free BSD documentation
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/

15. Free BSD handbook. Kernel building.
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig.html

16. SAMHAIN download site.
http://www.la-samhna.de/samhain/samhain-current.tar.gz

17. SAMHAIN documentation
http://www.la-samhna.de/samhain/manual/index.html

18. NMAP. Port scanner.
http://www.insecure.org/nmap/

19. Etherboot Homepage. Network boot tool.
www.etherboot.org

20. OpenSSL homepage. SSL/TLS libraries.
http://www.openssl.org/source/

21. Debian news forum.
http://www.debian.org/News/weekly/1999/45/

22. SAMHAIN manual. On stealth mode.
chttp://www.la-samhna.de/samhain/manual/stealthmode.html

23. Phrack magazine. Exploits.
http://www.phrack.org/archives/61/p610x03_Linenoise.txt.

Appendix A

System setup walk-through

In this appendix we will take you through the setup of the system in detail. To further document the setup phase and to make it easier to recreate we will add all changes made in configuration files in appendix B. It is important to note that we will be working as root (superuser) during the setup section and that all paths are given as absolute paths from the root of the directory tree. We have chosen not to comment on things clearly stated in the system *man pages* such as syntax and options, if you at any time wish to use different options or syntax please read the manual pages[1]

A.1 The setup of the Debian based file servers

We started out with a clean Debian installation. We will be using the aptitude package manager to install packages as this program includes all dependencies. We patched our Debian Sarge 3.2 with a 2.4 Kernel and the latest security patches for this kernel release.

```
aptitude update
```

We wished to use the Lessdisks repository as source for our Lessdisks installation and added the following to the */etc/apt/sources.list*

```
deb http://Lessdisks.net/Debian/current /
```

Later we discovered that the packages from the Lessdisks repository were corrupt, so we removed them and used the equivalent packages from the Debian repository[2].

In the setup phase we need direct Internet access to use the repositories, so we configure the NIC's by editing */etc/network/interfaces*. For the complete file, please read appendix B.1.1. We also configured the eth1 NIC as shown in figure 3.1.

[1]The manual pages for program xyz is found with the command *man xyz*

[2]We add this warning to our project to stop others from making the same mistake

Then we install the basic packages we need.

```
aptitude install Lessdisks locales most
```

We need the locales package as this package contains the internationalization files of for the *util-Linux* package which is used by Lessdisks. The *most* package is a useful paging program that displays, one window at the time, the contents of a file on a terminal. For more information on these programs please read their *man pages*. These are the basic packages needed for Lessdisks as stated in the Lessdisks documentation[3].

During the installation of Lessdisks you will be asked several questions to help you configure Lessdisks. We will mention some of the most important options here but for a full transcript of the options used, we refer you to appendix B It is recommended that you use *thy*, which is a http daemon, in combination with the Debian Sarge distribution. The file server should supply the kernel image so the option kernel-image-netbootable must be loaded. We also recommend that Lessdisks install the DHCP3 and syslinux packages. We will explain the importance of these later in this section.

During a test of the boot environment we discovered that PXE on our 3COM card did not support Network File System (NFS), but only TFTP. We therefore installed the package tftp-hpa. The tftp-hpa package is a TFTP server derived from Open-BSD TFTP with some extra options added and bugs fixed.

```
aptitude install  tftp-hpa
```

The use of PXE TFTP became a complicating factor for us since it deviated from the setup in [Nyr05] and posed a much more complex challenge. Our solution is to use TFTP to transfer the the Linux kernel and then use NFS to transfer the file system.

At boot, the PXE-client will start and will receive an IP from the DHCP server. It will also acquire the file containing the boot-loader and the service name of the tftp-server. This file, pxeboot.0 is located at */var/lib/lessdisks/boot/*. It is a small program with the same basic functionality as any boot-loader, e.g. lilo/grub (http://lilo.go.dyndns.org/). It is transferred via TFTP to the application server. Pxeboot.0 comes with the syslinux package and will need to be copied from */var/lib/syslinux/* to */var/lib/lessdisks/boot/*. If you choose not to let Lessdisks install the syslinux package then use the aptitude command to install the package. As always, you should accept to install the package dependencies. After coping the pxeboot.0 the system needs some additional configuration options for loading the kernel over the network. These options are contained in the file *default* and should be placed under */var/lib/lessdisks/boot/pxelinux.cfg*. This file

[3]The Lessdisks documentation is found at /usr/share/doc/Lessdisks and /usr/share/doc/Lessdisks-doc

contains the export parameters for the kernel. This directory does not exist so we need to create it before we copy the *default* file to this directory.

```
cd /var/lib/syslinux
cp pxeboot.0 /var/lib/lessdisks/boot/
cd /var/lib/lessdisks/boot/
mkdir pxelinux.cfg
cd /var/lib/syslinux/pxelinux
cp default /var/lib/lessdisks/boot/pxelinux.cfg
```

Pxeboot.0 will use the configuration files located in */var/lib/lessdisks/boot/pxelinux.cfg/default* to load and start the Linux core, vmlinuz. Vmlinuz is a compressed Linux kernel, which is bootable. Bootable in this context means that it is capable of loading the operating system into memory so that the computer becomes operational. The kernel is also transferred via TFTP. The Debian kernel will use UDHCPC to regain the IP used by pxeboot. Once it has regained the IP the application server will load the file system over NFS and boot like a normal computer.

In order to archive this setup we must do a number of things. First we need to configure DHCP to provide an IP address to the application server at booting. DHCP will also supply information about where to find the kernel image, gateway and the MPITS system which is functioning as a router. If you chose the DHCP package under the Lessdisks install, then DHCP3 will be installed on your computer. If you ignored the DHCP option then you must install it manually To configure DHCP we edit the file dhcpd.conf in the directory */etc/dhcp3/*. A copy of the file can be found in B.1.3 in appendix B.

```
aptitude install  dhcp3
```

When we tested the application server to see if DHCP was working we got an error message about a *spurious 8259A interrupt: IRQ7*. After googling this error message and reading several forums we found out that there is an driver bug in the 3com NIC in the 2.4 and 2.6 kernel. As all our NIC‘s are 3COM cards, we had to apply a fix found on the web. The fix was originally proposed by Peter Rundle in[13].

We have included the vital parts of the dhcpd.conf in appendix B. The key elements consists of allowing network booting, defining the subnet, IP and external router for the application server and the address and file path to the *next-server* which in this case is the file server. We also here apply the fix that makes the 3COM NIC work, the fix is clearly marked in section B.1.3.

Now that DHCP is working we have to configure TFTP to deliver the kernel image to the application server. First we edit the Internet server configuration database, */etc/inetd.conf*

```
#:BOOT: Tftp service is provided primarily for booting.
# most sites run this only on machines acting as ‘‘boot servers’’

tftp dgram upd wait root /usr/sbin/tcpd /usr/sbin/in.tftpd /var/lib/lessdisks/boot
```

The first part gives the service name, socket type, and transport protocol. The last part contains flags, the user, and the path to the service and the kernel image for the application server.

We also have to set the access control rights for what we allow the system to export. We do this in */etc/exports*.

```
# /etc/exports

/var/lib/Lessdisks 10.2.0.1(rw,no_root_squash,synch)
```

The rw, read/write, option gives rw access to the exported file system. *root_squash*, *no_root_squash* or *all_squash* specifies whether the root user of the client should be squashed, i.e. that the User ID (UID) in the system is mapped to the user *nobody*. *all_squash* will map all users to the user *nobody*. The *sync/async* determines whether NFS should be able to reply to requests before committing changes to disk. *async* allows this, *sync* does not.

Lessdisks utilizes UDHCP which is a small version of DHCP meant for embedded systems. For some reason, when it uses UDHCP it crashes with the following message

```
WARNING: udhcp bug workaround, setting dhcp server to next-server()
attempting to mount NFS file system...
mount -nt nfs -o rw 10.2.0.2:/var/lib/lessdisks /mnt
mount: RPC: Remote system error- Network is unreachable
/bin/sh: can't access tty; job control turned off
\#spurious $8259A$ interrupt: IRQ$7$
```

It seems like when the lessdisks kernel loads it uses some sort of bug workaround, but the kernel does not't realize that it should contact the file server / *next-server* to obtain it's IP. After reading the bash script files at */etc/lessdisks/mkinitrd/install_scripts/* we did not succeed in finding the error in the scripts. But we realized which variable we needed to set to make this work. In the file 70_dhcp we assigned the value "dhcp" to the variable autoconf_type. We have included the file in Appendix B. Then we recompiled the lessdisks kernel and actually managed to get the application server to do a complete boot. To rebuild the kernel image we used the command:

```
lessdisks-chroot dpkg-reconfigure kernel-image-2.4.27-3-386
```

To make Lessdisks function properly with NFS we found that we must also install the *portmap* package both on the file and application server. Portmap converts RPC program number into DARPA protocol port numbers and must be running in order to make RPC calls.

```
aptitude install portmap
lessdisks-aptget install portmap
```

A.2 The setup of the Lessdisks Debian based Application server

Since we use PXE all we need to configure on application servers is the bios and the NIC. The BIOS must allow booting via the NIC and the NIC must be configured to use PXEboot. This is done by accessing the bios, choosing network as the preferred boot environment and disabling all others. Subsequently you enter the NIC's configuration by pressing keys *Alt + Ctrl + B* when the NIC boot message is displayed. Here you enable the PXE boot option. The rest of the configuration for the boot process is done on the server side. Once the application server boots we must configure the NIC of the application server by editing */etc/network/interfaces*. Here we enable the data interface towards the MPITS. We have included a copy of the interfaces file in appendix B under B.2.1.

We then installed Apache web server on the application servers according to table 3.3 by using the Lessdisks apt get command. Since we have one server running Apache 2 and one running Apache 1.3 please exchange "package" below with the Apache package you install.

```
lessdisks-aptget install package
```

A.2.1 Setup of Apache 1.3

First we edit the httpd.conf with address, port, log and document root information. The document root is by default located at */var/www/* on a Debian system. Here we place our HTML document and use *chmod 0644 file* to set the access rights to the file.

A.2.2 Setup of Apache 2

After installing Apache 2, we edit the */etc/apache2/apache.conf* file with the necessary options. Then we make a file in the */etc/apache2/sites-available* directory. This will contain meta information about our web resources. The file is shown below:

```
<VirtualHost *>
ServerName appserver1
DocumentRoot /var/www/pro/
Options FollowSymLinks
</VirtualHost>
```

Then we enable the web site by using the *a2ensite* command followed by the name of the meta data file created earlier. Finally we create the HTML page for our system and place it in */var/www/pro*.

A.3 The setup of the FreeBSD based file servers

To get Internet access with FreeBSD version 5.4 we had to configure the interfaces xl0 and xl1 in /etc/rc.conf with IPaddresses, subnet and gateway information. We also make sure that the NFS and DHCP server options are enabled. A copy of the file can be found in the appendix B under B.3.1. It is important to notice that */etc/rc.conf* is an override file for */etc/defaults/rc.conf* which contains all options with default settings. We use the Free BSD ports collection to install all needed services and programs that are available through this collection.

To install DHCP we use the ISC DHCP software package in the ports collection. It is located at */usr/ports/net/isc-dhcp3-server/*. At this location we find a makefile and run *make* and *make install*. When running make we selected the basic installation option with no extra functionality.

Then we proceeded to edit */usr/local/etc/dhcpd.conf*. We include boot information about the pxebootloader in the file name option as well at the rootpath for the new application server. DHCP will deliver this information to the application server as a reply to the pxe clients DHCP REQUEST. We refer to figure 3.1 and 3.4 and table 3.1 for more information on the address configuration and the boot process. A copy of this file is in appendix B under B.3.2.

We need to enable TFTP. TFTP is installed as default in Free BSD, all we have to do is start the TFTP daemon by configuring */etc/inetd.conf*. We set the path so that when the pxe client on the application servers NIC asks for the file, pxeboot, TFTP will know where to find it.

```
tftp dgram udp wait root /usr/libexec/tftpd tftpd -l -s /usr/diskless/pxeboot
```

The option INETD_ENABLE must be set in the */etc/rc.conf* file for this command to execute correctly. The -l switch turns on the logging of TFTP operations. The -s switch specifies the root directory for tftpd after it calls chroot(). For more information on tftpd switches, please read the TFTP man page.

Now that the support services are configured, we need to build a diskless kernel for the application server. First, we need to install the kernel source. We install the source from the FreeBSD CD.

```
mount /cdrom
mkdir -p /usr/src/sys
cd /usr/src/sys
ln -s /usr/src/sys /sys
cat /cdrom/src/ssys.[a-d]* |tar -xzvf
```

Then we must create the diskless directory, which will be the root fore the file system for the diskless server. The option -p in mkdir allows intermediate directories to be created as required. Intermediate directories are created with the permission bits of *rwxrwxrwx* (0777).

```
cd /usr
mkdir -p diskless
cd /usr/src
make buildworld DESTDIR=/usr/diskless
make installworld DESTDIR=/usr/diskless
```

We name the directory diskless, and populate the new root by running the make world command in */usr/src*. This creates a new file system inside the diskless directory as seen below. Be warned that this is a very time consuming process that can take several hours.

```
bin dev lib mnt pxeboot root sys usr
boot etc libexec proc rescue sbin tmp var
```

Then we need to copy the pxebootloader from */boot* to the diskless application server boot directory, */diskless/boot*. We now TFTP will be able to deliver the pxeboot file to the booting pxe client.

```
cd /usr/diskless
mkdir pxeboot
```

```
cp /boot/pxeboot pxeboot
```

We have earlier enabled the NFS to start during the file servers booting procedure. Now we need to configure NFS to specify which path should be exported to the application server and set the access rights. We do this in the */etc/exports* file by the command:

```
echo "/usr/diskless -mapall=root 10.3.0.1" >> /etc/exports
```

The CPU on the diskless client is an AMD 32 bit processor so we use a i386 generic kernel as a basis for our diskless kernel. The kernel is found in */usr/src/sys/i386/conf* and is named GENERIC. We need to adapt the kernel configuration to our diskless server. You want to keep the GENERIC kernel configuration intact, since it is quite an error prone process to build a kernel and it is likely that you will wish to start over fresh during the build. So first we make a copy of the GENERIC file, put it somewhere, and create a symlink, symbolic link to the file from the */usr/src/sys/i386/conf* directory.

```
cd /usr/src/sys/i386/conf
cp GENERIC /diskless/boot/kernels/APPSERVER
ln -s /diskless/boot/kernels/APPSERVER
```

Now we edit the APPSERVER file with an text editor of choice. This is a general configuration file, where each line contains a keyword and one or more arguments. Anything following a # is considered a comment. The file is divided into kernel options, arguments and device specific information. Making a diskless kernel compile is a game of trial and error. You have to know the hardware specifics of the machine you are working on as well the internal dependencies between options and devices. We have included our APPSERVER file in appendix B under B.3.3. We have commented out parts of the device list to make it easy to adapt this diskless kernel to another set of hardware and we have deleted configuration lines containing options not suited for diskless nodes.

After editing the file we use the configure command to load our selected modules into the */usr/src/sys/i386/compile/APPSERVER* directory. Here we run make depend to make sure we have included all dependencies.

```
cd /usr/src/sys/i386/conf/
configure APPSERVER
cd ../compile/APPSERVER/
make depend
```

Now we build the kernel, working from the */usr/src/* directory. The kernel will install it self plus the needed kernel modules under */usr/diskless/boot/kernel/* and the kernel source under */usr/src/sys/*. The last make command copies the configuration files from

the */etc/* directory and installs them on the appropriate place.

```
cd /usr/src
make buildkernel KERNEL=APPSERVER DESTDIR=/usr/diskless
make installkernel KERNEL=APPSERVER DESTDIR=/usr/diskless
cd /usr/src/etc
make distribution DESTDIR=/usr/diskless
```

A.4 Setup of the Free BSD based Application server

We need to configure the BIOS and NIC in the same way as we did in section A.2. Then we must configure */etc/rc.conf*. We must enable the NFS client option and configure the other NIC with a static IPaddress. You can find a copy of the application server's */etc/rc.conf* file in appendix B under B.4.1.

After we have configured the NICs we install Apache. The Apache package can be found in the Free BSD ports collection under */usr/ports/www/apache13/*. To install Apache 1.3 on Free BSD we must specify the destination directory. All other install options are default. We run make and make install and start Apache.

```
./configure --prefix=/usr/local/apache
     $ make
     $ make install
     $ /usr/local/apache/bin/apachectl start
```

We configure */usr/local/apache/conf/httpd.conf* with listening ports, IP address, log and web resource path. And finally we create the static web page content and place the HTML document in the */usr/local/apache/htdocs/* directory.

A.5 Setup of the SAMHAIN client

Now we must set up the SAMHAIN client on the file servers. The setup procedure is identical for both Debian and Free BSD and we will here use the generic term operating system when referring to the underlying system. First we must download SAMHAIN from (http://www.la-samhna.de/samhain/samhain-current.tar.gz). Then extract it, verify the Pretty Good Privacy (PGP) signature and run the configure, make and make install commands.

```
cd /usr/local

gunzip -c samhain-current.tar.gz | tar xvf -
gpg --verify samhain-2.3.0a.tar.gz.asc samhain-2.3.0a.tar
unzip samhain-2.3.0a.tar.gz | tar xvf -
```

```
cd samhain-2.3.0a

./configure --enable-network=client ---disable-encrypt

make

make install
```

After installing SAMHAIN we configure the file */etc/samhainrc*. This file contains a set of predefined policies which are mapped to the files we wish to monitor. We will add the *samhainrc* file appendix B in section B.1.5. We must also set the IP address of the Yule log server. The SetLogServer entry can be found under

Misc

in the samhainrc configfile which is inculded in appendix B section B.1.5. We will then initialize the baseline database which contains all the file signatures and then we run a check on the system enabling the deamon functionality.

```
samhain -t init
samhain -t check -D --forever
```

For more information on SAMHAIN we refer to the SAMHAIN man page.

A.6 Setup of the MPITS

We run the MPITS on a Debian Sarge Linux distribution. To configure the NIC with IP addresses we configure */etc/network/interfaces* as seen in table 3.1. The configuration file can be found in appendix B under B.5.1. To install the MPITS we mount the CDROM and copy the files to the hard drive. Afterwards we must compile the C files before running the MPITS. To compile the C code any C compiler we do, we use gcc, the GNU C Compiler

```
mount /cdrom
cd /cdrom/Proxy
cp -r . /home/usr/Proxy
gcc ccodefile.c -o ccondefile
```

Then we must configure the MPITS to our network topology. This is done by editing the mpits.conf file found at /home/usr/Proxy. Here we set the application server's IP addresses so the MPITS know which IP addresses to forward the request to. Then we can start the MPITS by running the compiled source code.

```
cd /pathToCompiledCode/
./proxy
```

All status and log messages will be printed on screen.

A.7 Setup of SAMHAIN log server

First we must download SAMHAIN from [6]. Then extract it, verify the PGP signature and run the *configure* with the options seen below, *make* and *make install* commands. By default, the *./configure* will check for the existence of a user Yule, daemon, or the user nobody, and use the first match. We override this by mapping the user to our system user account.

```
cd /usr/local

gunzip -c samhain-current.tar.gz | tar xvf -
gpg --verify samhain-2.3.0a.tar.gz.asc samhain-2.3.0a.tar
unzip samhain-2.3.0a.tar.gz | tar xvf -

cd samhain-2.3.0a

./configure --enable-network=server --enable-identity=jantore --disable-encrypt

make

make install
```

Appendix B

Configuration files

B.1 The configuration files for the Debian fileserver setup

B.1.1 /etc/network/interfaces

```
# /etc/network/interfaces

# the loopback network interface
auto lo
iface lo inet loopback

# the primary interface
auto eth0
iface eth0 inet static
address 129.241.252.122
netmask 255.255.255.240
network 129.241.252.112
broadcast 129.241.252.127
gateway 129.241.252.113

# the DNS
dns-nameservers 129.241.190.190

# The fileserver network interface
auto eth1 inet static
address 10.2.0.2
netmask 255.255.255.0
network 10.2.0.0
broadcast 10.2.0.255
```

B.1.2 Lessdisk configuration values

```
lessdisks_admins=''jantore''
packages=''lessdisks-xterminal less vim udhcpc discover-data more''
kernel_packages=''kernel-image-bootable''
lessdisksarchive=''''
defaultarchive=''http.us.debian.org/''
archivenonus=''non-us.debian.org/''
archivesecurity=''security.debian.org/''
http_proxy=''''
newarch=''i386''
disk_server=''filserver2''
lessdisks_path=''var/liv/lessdisks''
disk_alias=''disk''
use_mknbi=''false''
mknbi_opts=''--ip=dhcp''
lessdisks_group=''lessdisks''
group_only=''yes''
rwfilesystem=''tmpfs''
rw=''var/state/lessdisks''
rw_dirs=''/tmp /var/log /var/spool /var/run /var/lock /var/cache/man
/var/lib/ntp /var/lib/xkb /var/lib/dhcp /var/tmp''
copy_dirs=''''
tmpfs_size=''10m''
floppy=''true''
cdrom=''true''
export_type=''thy''
aptget_binary=''apt-get''
lessdisks_frontend=''noninteractive''
debian_dist=''sarge''
debootstrap=''usr/sbin/debootstrap''
debootstrap=_opts=''''
exclude_packages=''lilo ipchains pcmcia-cs ppp pppoe pppconfig exim emix4-base
exim4 exim4-config exim4daemon-light logrotate mailx at''
include_paclages=''''
```

B.1.3 DHCPD.conf

```
# /etc/dhcp3/dhcpd.conf
# DHCP Configuration file

option domain-name-servers 129.241.190.190 ;
ddns-update-style none;

default-lease-time 600;
max-lease-time 7200;

# If the DHCP server is the official DHCP server for the local network,
# the authorative directive should be uncommented.
authorative;

allow booting;

subnet 10.2.0.0 netmask 255.255.255.0 {
        # range dynamic-bootp 10.2.0.1 10.2.0.1;
        #use-host-decl-names on;
        option broadcast-address 10.2.0.255;
        option routers 10.0.0.4;

        host appserver2{
                # filename "vmlinuz.nb"
                hardware ethernet 00:0a:5e:04:65:a5;
                fixed-address 10.2.0.1;
                option root-path "10.2.0.2:/var/lib/lessdisks/";
                next-server 10.2.0.2;
                option host-name "appserver2";

                #trick from Peter Rundle <peter.rundle@au.interpath.net>
                if substring (option vendor-class-identifier, 0, 9)= "PXEClient"
                {
                        filename "/var/lib/lessdisks/boot/pxelinux.0";
                        # NOTE : Kernels are spesified in boot/pxelinux.cfg/
                }
                else
                {
                        filename "/var/lib/lessdisks/boot/vmlinuz";
                }
        }
}
```

B.1.4 70_DHCP

```
#!/bin/sh

. /etc/lessdisks/mkinitrd/initrd-netboot.conf

if [ -r ‘‘$cmdline_opts" ]; then
. ‘‘$cmdline_opts’’
fi

if [ ‘‘/dev/nfs" !=$rootdev]; then
echo ‘‘root not set to /dev/nfs, skipping dhcprequest...’’
exit 0;;
fi

case $atuoconf_type in
off|none) echo ‘‘not configuring ip address"
exit 0 ;;
dhcp|bootp|both) ;;
*) ;;
esac

if [ -z ‘‘$(/sin/ifconfig eth0)" ]; then
echo ‘‘not configuring DHCP, no ethernet card found...’’
exit 10
fi

network_script=/etc/lessdisks/mkinitrd/network_script

getDhclient() {
/sbin/dhclient

#kill dhclient so the initrd can get un-mounted
killall dhclient 2> /dev/null || killall dhclient-2.2.x 2> /dev/null
}

getUdhcpc(){
/sbin/udhcpc --forground --quit --script=$network_script
}
```

```
getCmdline (){
. ‘‘$cmdline_opts’’
test -z ‘‘$interface’’ && interface=’’eth0’’
export interface
if [  ‘‘$1’’=’’runscript’’ ]; then
$network_script bound
fi
}

getCmdline

## We assign the value dhcp to the variable autoconf_type since it
## for some reason doesn’t get this automatically.

autoconf_type=dhcp

if[ -n ‘‘$ip" ]; then
getCmdline runscript
else
case $autoconf_type in
dhcp|bootp)
if [ -x /sbin/udhcpc ]; then
echo ‘‘attempting to configure DHCP with udhcpc’’
getUdhcpc
elif [-x /sbin/dhclient ]; then
echo ‘‘attempting to configure DHCP with dhclient’’
getDhclient
else
echo ‘‘no DHCP client found, attempting to get values from /proc/cmdline’’
fi
;;
*) getCmdline runscript ;;
esac
fi
```

B.1.5 The samhainrc file

```
######################################################################
#
# Configuration file template for samhain.
#
######################################################################
#
# -- empty lines and lines starting with '#', ';' or '//' are ignored
# -- boolean options can be Yes/No or True/False or 1/0
# -- you can PGP clearsign this file -- samhain will check (if compiled
#    with support) or otherwise ignore the signature
# -- CHECK mail address
#
# To each log facility, you can assign a threshold severity. Only
# reports with at least the threshold severity will be logged
# to the respective facility (even further below).
#
######################################################################
#
# SETUP for file system checking:
#
# (i)   There are several policies, each has its own section. Put files
#       into the section for the appropriate policy (see below).
# (ii)  Section [EventSeverity]:
#       To each policy, you can assign a severity (further below).
# (iii) Section [Log]:
#       To each log facility, you can assign a threshold severity. Only
#       reports with at least the threshold severity will be logged
#       to the respective facility (even further below).
#
######################################################################

######################################################################
#
# Files are defined with: file = /absolute/path
#
# Directories are defined with:                   dir = /absolute/path
# or with an optional recursion depth (N <= 99): dir = N/absolute/path
#
# Directory inodes are checked. If you only want to check files
# in a directory, but not the directory inode itself, use (e.g.):
#
```

```
# [ReadOnly]
# dir = /some/directory
# [IgnoreAll]
# file = /some/directory
#
# You can use shell-style globbing patterns, like: file = /path/foo*
#
#########################################################################

[Misc]
##
## Add or subtract tests from the policies
## - if you want to change their definitions,
##   you need to do that before using the policies
##
# RedefReadOnly = (no default)
# RedefAttributes=(no default)
# RedefLogFiles=(no default)
# RedefGrowingLogFiles=(no default)
# RedefIgnoreAll=(no default)
# RedefIgnoreNone=(no default)
# RedefUser0=(no default)
# RedefUser1=(no default)

[Attributes]
##
## for these files, only changes in permissions and ownership are checked
##
file=/var/lib/lessdisks/etc/mtab
file=/var/lib/lessdisks/etc/ssh_random_seed  #File does not exist
file=/var/lib/lessdisks/etc/asound.conf #File does not exist
file=/var/lib/lessdisks/etc/resolv.conf
file=/var/lib/lessdisks/etc/localtime
file=/var/lib/lessdisks/etc/ioctl.save #File does not exist
file=/var/lib/lessdisks/etc/passwd.backup #File does not exist
file=/var/lib/lessdisks/etc/shadow.backup #File does not exist

#
# There are files in /etc that might change, thus changing the directory
# timestamps. Put it here as 'file', and in the ReadOnly section as 'dir'.
#
file=/var/lib/lessdisks/etc
```

```
[LogFiles]
##
## for these files, changes in signature, timestamps, and size are ignored
##
file=/var/lib/lessdisks/var/run/utmp
file=/var/lib/lessdisks/etc/motd



########################################################################
#
# This would be the proper syntax for parts that should only be
#    included for certain hosts.
# You may enclose anything in a @HOSTNAME/@end bracket, as long as the
#    result still has the proper syntax for the config file.
# You may have any number of @HOSTNAME/@end brackets.
# HOSTNAME should be the fully qualified 'official' name
#    (e.g. 'nixon.watergate.com', not 'nixon'), no aliases.
#    No IP number - except if samhain cannot determine the
#    fully qualified hostname.
#
# @HOSTNAME
# file=/foo/bar
# @end
#
# These are two examples for conditional inclusion/exclusion
# of a machine based on the output from 'uname -srm'
#
# $Linux:2.*.7:i666
# file=/foo/bar3
# $end
#
# !$Linux:2.*.7:i686
# file=/foo/bar2
# $end
#
########################################################################

[GrowingLogFiles]
##
## for these files, changes in signature, timestamps, and increase in size
##                  are ignored
##
```

```
file=/var/lib/lessdisks/var/log/warn #Does not exist
file=/var/lib/lessdisks/var/log/messages #Does not exist
file=/var/lib/lessdisks/var/log/wtmp
file=/var/lib/lessdisks/var/log/faillog #Does not exist
file=/var/lib/lessdisks/var/log/auth.log
file=/var/lib/lessdisks/var/log/daemon.log #Does not exist
file=/var/lib/lessdisks/var/log/user.log #Does not exist
file=/var/lib/lessdisks/var/log/kern.log #Does not exist
file=/var/lib/lessdisks/var/log/syslog #Does not exist


[IgnoreAll]
##
## for these files, no modifications are reported
##
## This file might be created or removed by the system sometimes.
##
file=/var/lib/lessdisks/etc/resolv.conf.pcmcia.save


[IgnoreNone]
##
## for these files, all modifications (even access time) are reported
##    - you may create some interesting-looking file (like /etc/safe_passwd),
##      just to watch whether someone will access it ...
##
file=/var/lib/lessdisks/etc/passwd_all_safe
file=/var/lib/lessdisks/var/www/index.html

[Prelink]
##
## Use for prelinked files or directories holding them
##


[ReadOnly]
##
## for these files, only access time is ignored
##
dir=/var/lib/lessdisks/usr/bin
dir=/var/lib/lessdisks/bin
dir=/var/lib/lessdisks/boot
#
```

```
# SuSE (old) has the boot init scripts in /sbin/init.d/*,
# so we go 3 levels deep
#
dir=3/var/lib/lessdisks/sbin
dir=/var/lib/lessdisks/usr/sbin
dir=/var/lib/lessdisks/lib
#
# RedHat and Debian have the bootinit scripts in /etc/init.d/* or /etc/rc.d/*,
#        so we go 3 levels deep there too
#
dir=3/var/lib/lessdisks/etc

# Various directories / files that may include / be SUID/SGID binaries
#
#
dir=/var/lib/lessdisks/usr/X11R6/bin
#dir=/usr/X11R6/lib/X11/xmcd/bin
file=/var/lib/lessdisks/usr/lib/pt_chown
#dir=/opt/gnome/bin
#dir=/opt/kde/bin

[User0]
[User1]
## User0 and User1 are sections for files/dirs with user-definable checking
## (see the manual)


[EventSeverity]
##
## Here you can assign severities to policy violations.
## If this severity exceeds the treshold of a log facility (see below),
## a policy violation will be logged to that facility.
##
## Severity for verification failures.
##
# SeverityReadOnly=crit
# SeverityLogFiles=crit
# SeverityGrowingLogs=crit
# SeverityIgnoreNone=crit
# SeverityAttributes=crit
# SeverityUser0=crit
# SeverityUser1=crit
```

```
# Default behaviour
SeverityReadOnly=crit
SeverityLogFiles=crit
SeverityGrowingLogs=warn
SeverityIgnoreNone=crit
SeverityAttributes=crit


##
## We have a file in IgnoreAll that might or might not be present.
## Setting the severity to 'info' prevents messages about deleted/new file.
##
# SeverityIgnoreAll=crit
SeverityIgnoreAll=info

## Files : file access problems
SeverityFiles=crit

## Dirs  : directory access problems
SeverityDirs=crit

## Names : suspect (non-printable) characters in a pathname
SeverityNames=crit

# Default behaviour
SeverityFiles=crit
SeverityDirs=crit
SeverityNames=warn


[Log]
##
## Switch on/OFF log facilities and set their threshold severity
##
## Values: debug, info, notice, warn, mark, err, crit, alert, none.
## 'mark' is used for timestamps.
##
##
## Use 'none' to SWITCH OFF a log facility
##
## By default, everything equal to and above the threshold is logged.
## The specifiers '*', '!', and '=' are interpreted as
## 'all', 'all but', and 'only', respectively (like syslogd(8) does,
```

```
## at least on Linux). Examples:
## MailSeverity=*
## MailSeverity=!warn
## MailSeverity==crit

## E-mail
##
# MailSeverity=none

## Console
##
# PrintSeverity=info

## Logfile
##
# LogSeverity=mark

## Syslog
##
# SyslogSeverity=none

## Remote server (yule)
##
ExportSeverity=crit

## External script or program
##
# ExternalSeverity = none

## Logging to a database
##
# DatabaseSeverity = none

# Default behaviour
MailSeverity=crit
PrintSeverity=none
LogSeverity=info
SyslogSeverity=alert
ExportSeverity=none
```

```
#########################################################
#
# Optional modules
#
#########################################################

# [SuidCheck]
##
## --- Check the filesystem for SUID/SGID binaries
##

## Switch on
#
# SuidCheckActive = yes

## Interval for check (seconds)
#
# SuidCheckInterval = 7200

## Alternative: crontab-like schedule
#
# SuidCheckSchedule = NULL

## Directory to exclude
#
# SuidCheckExclude = NULL

## Limit on files per second (0 == no limit)
#
# SuidCheckFps = 0

## Alternative: yield after every file
#
# SuidCheckYield = no

## Severity of a detection
#
# SeveritySuidCheck = crit

## Quarantine SUID/SGID files if found
#
# SuidCheckQuarantineFiles = yes
```

```
## Method for Quarantining files:
#  0 - Delete or truncate the file.
#  1 - Remove SUID/SGID permissions from file.
#  2 - Move SUID/SGID file to quarantine dir.
#
# SuidCheckQuarantineMethod = 0

## For method 1 and 3, really delete instead of truncating
#
# SuidCheckQuarantineDelete = yes

[Kernel]
##
## --- Check for loadable kernel module rootkits (Linux/FreeBSD only)
##

## Switch on/off
#
KernelCheckActive = True

## Check interval (seconds); btw., the check is VERY fast
#
KernelCheckInterval = 300

## Severity
#
SeverityKernel = crit


[Utmp]
##
## --- Logging of login/logout events
##

## Switch on/off
#
LoginCheckActive = True

## Severity for logins, multiple logins, logouts
#
SeverityLogin=info
SeverityLoginMulti=warn
```

```
SeverityLogout=info

## Interval for login/logout checks
#
LoginCheckInterval = 300


# [Database]
##
## --- Logging to a relational database
##

## Database name
#
# SetDBName = samhain

## Database table
#
# SetDBTable = log

## Database user
#
# SetDBUser = samhain

## Database password
#
# SetDBPassword = (default: none)

## Database host
#
# SetDBHost = localhost

## Log the server timestamp for received messages
#
# SetDBServerTstamp = True

## Use a persistent connection
#
# UsePersistent = True

# [External]
##
## Interface to call external scripts/programs for logging
```

```
##

## The absolute path to the command
## - Each invocation of this directive will end the definition of the
##   preceding command, and start the definition of
##   an additional, new command
#
# OpenCommand = (no default)

## Type (log or rv)
## - log for log messages, srv for messages received by the server
#
# SetType = log

## The command (full command line) to execute
#
# SetCommandLine = (no default)

## The environment (KEY=value; repeat for more)
#
# SetEnviron = TZ=(your timezone)

## The TIGER192 checksum (optional)
#
# SetChecksum = (no default)

## User who runs the command
#
# SetCredentials = (default: samhain process uid)

## Words not allowed in message
#
# SetFilterNot = (none)

## Words required (ALL of them)
#
# SetFilterAnd = (none)

## Words required (at least one)
#
# SetFilterOr = (none)

## Deadtime between consecutive calls
```

```
#
# SetDeadtime = 0

## Add default environment (HOME, PATH, SHELL)
#
# SetDefault = no


#########################################################
#
# Miscellaneous configuration options
#
#########################################################

[Misc]

[Misc]

SetLogServer=10.0.1.4
## whether to become a daemon process
## (this is not honoured on database initialisation)
#
# Daemon = no
Daemon = yes

## whether to test signature of files (init/check/none)
## - if 'none', then we have to decide this on the command line -
#
# ChecksumTest = none
ChecksumTest=check

## whether to drop linux capabilities that are not required
## - will make a root process a 'mere mortal' in many respects
#
# UseCaps = yes

## Set nice level (-19 to 19, see 'man nice'),
## and I/O limit (kilobytes per second; 0 == off)
## to reduce load on host.
#
# SetNiceLevel = 0
# SetIOLimit = 0
```

```
## The version string to embed in file signature databases
#
# VersionString = NULL

## Interval between time stamp messages
#
# SetLoopTime = 60
SetLoopTime = 600

## Interval between file checks
#
# SetFileCheckTime = 600
SetFileCheckTime = 60

## Alternative: crontab-like schedule
#
# FileCheckScheduleOne = NULL

## Alternative: crontab-like schedule(2)
#
# FileCheckScheduleTwo = NULL

## Report only once on modified fles
## Setting this to 'FALSE' will generate a report for any policy
## violation (old and new ones) each time the daemon checks the file system.
#
# ReportOnlyOnce = True

## Report in full detail
#
ReportFullDetail = True

## Report file timestamps in local time rather than GMT
#
UseLocalTime = Yes

## The console device (can also be a file or named pipe)
## - There are two console devices. Accordingly, you can use
##   this directive a second time to set the second console device.
##   If you have not defined the second device at compile time,
##   and you don't want to use it, then:
##   setting it to /dev/null is less effective than just leaving
##   it alone (setting to /dev/null will waste time by opening
```

```
##   /dev/null and writing to it)
#
# SetConsole = /dev/console

## Activate the SysV IPC message queue
#
# MessageQueueActive = False


## If false, skip reverse lookup when connecting to a host known
## by name rather than IP address (i.e. trust the DNS)
#
# SetReverseLookup = True

## --- E-Mail ---

# Only highest-level (alert) reports will be mailed immediately,
# others will be queued. Here you can define, when the queue will
# be flushed (Note: the queue is automatically flushed after
# completing a file check).
#
SetMailTime = 120

## Maximum number of mails to queue
#
SetMailNum = 10

## Recipient (max. 8)
#
SetMailAddress=root@10.0.1.4
setMailAddress=root@localhost

## Mail relay (IP address)
#
# SetMailRelay = NULL

## Custom subject format
#
MailSubject = [Samhain at %H] %T: %S

## --- end E-Mail ---

## Path to the prelink executable
```

```
#
# SetPrelinkPath = /usr/sbin/prelink

## TIGER192 checksum of the prelink executable
#
# SetPrelinkChecksum = (no default)


## Path to the executable. If set, will be checksummed after startup
## and before exit.
#
# SamhainPath = (no default)


## The IP address of the log server
#
# SetLogServer = (default: compiled-in)

## The IP address of the time server
#
# SetTimeServer = (default: compiled-in)

## Trusted Users (comma delimited list of user names)
#
# TrustedUser = (no default; this adds to the compiled-in list)

## Path to the file signature database
#
# SetDatabasePath = (default: compiled-in)

## Path to the log file
#
# SetLogfilePath = (default: compiled-in)

## Path to the PID file
#
# SetLockPath = (default: compiled-in)


## The digest/checksum/hash algorithm
#
# DigestAlgo = TIGER192
```

```
## Custom format for message header.
## CAREFUL if you use XML logfile format.
##
## %S severity
## %T timestamp
## %C class
##
## %F source file
## %L source line
#
# MessageHeader="%S %T "


## Don't log path to config/database file on startup
#
# HideSetup = False

## The syslog facility, if you log to syslog
#
# SyslogFacility = LOG_AUTHPRIV
SyslogFacility=LOG_LOCAL2

## The message authentication method
## - If you change this, you *must* change it
##   on client *and* server
#
# MACType = HMAC-TIGER


## everything below is ignored
[EOF]

########################################################################
# This would be the proper syntax for parts that should only be
#    included for certain hosts.
# You may enclose anything in a @HOSTNAME/@end bracket, as long as the
#    result still has the proper syntax for the config file.
# You may have any number of @HOSTNAME/@end brackets.
# HOSTNAME should be the fully qualified 'official' name
#    (e.g. 'nixon.watergate.com', not 'nixon'), no aliases.
#    No IP number - except if samhain cannot determine the
#    fully qualified hostname.
```

```
#
# @HOSTNAME
# file=/foo/bar
# @end
#
# These are two examples for conditional inclusion/exclusion
# of a machine based on the output from 'uname -srm'
# $Linux:2.*.7:i666
# file=/foo/bar3
# $end
#
# !$Linux:2.*.7:i686
# file=/foo/bar2
# $end
#
##############################################################################
```

B.2 THe configuration files for the Lessdisks Debian based Application server

B.2.1 etc/network/interfaces

```
# Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
# /usr/shareTdoc/ifupdown/examples for more information.

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
address 10.0.0.1
netmask
```

B.3 The configuration files for the FreeBSD setup

B.3.1 etc/rc.conf

```
# --sysinstall genetated deltas -- # Sun Oct 1 18:16:33 2006
# Created: Sun Oct 1 18:16:33 2006
# Enable network daemons for user convenience.
# Please make all changes to this file, not to /etc/defaults/rc.conf
# This file contains just the overrides from /etc/defaults/rc.conf.
hostname=''fileserver3''
ifconfig_xl0=''inet 129.241.252.121 netmask 255.255.255.240''
ifconfig_xl1=''inet 10.3.0.2 netmask 255.255.255.0''
defaultrouter=''129.241.252.113''
inetd_enable=''YES''
keymap=''norwegian.iso''
linux_enable=''YES''
moused_enabled=''YES''
nfs_server_enable=''YES''
rpcbind_enable=''YES''
usbd_enable=''YES''
dhcpd_enable=''YES''
```

B.3.2 /usr/local/etc/dhcpd.conf

```
# dhcpd.conf
#
default-lease-time 600;
max-lease-time 7200;
authoritative;
ddns-update-style none;

log-facility local7;

option domain-name-servers 129.241.190.190;
option routers 10.0.0.4;

subnet 10.0.0.3 netmask 255.255.255.0 {

}

subnet 10.3.0.0 netmask 255.255.255.0 {
use-host-decl-names on;
option subnet-mask 255.255.255.0;
option broadcast-address 10.3.0.255;
host appserver3 {
    hardware ethernet 00:04:76:25:98:f9;
     fixed-address 10.3.0.1;
     filename "/data/misc/kernel.diskless";
     option root-path "10.3.0.2:/data/misc/diskless";
   }
}

next-server 10.3.0.2;
```

B.3.3 /usr/src/sys/i386/conf/APPSERVER

```
#
# APPSERVER Configuration file
#


machine i386
cpu I486_CPU
cpu I586_CPU
cpu I686_CPU
ident APPSERVER

# To statically compile in device wiring instead of /boot/device.hints
#hints "GENERIC.hints" # Default places to look for devices.

options  SCHED_4BSD # 4BSD scheduler
options  INET # InterNETworking
options  INET6 # IPv6 communications protocols
options  FFS # Berkeley Fast Filesystem
options  SOFTUPDATES # Enable FFS soft updates support
options  UFS_ACL # Support for access control lists
options  UFS_DIRHASH # Improve performance on big directories
options  MD_ROOT # MD is a potential root device
options  NFSCLIENT # Network Filesystem Client
options  NFS_ROOT # NFS usable as /, requires NFSCLIENT
options  MSDOSFS # MSDOS Filesystem
options  CD9660 # ISO 9660 Filesystem
options  PROCFS # Process filesystem (requires PSEUDOFS)
options  PSEUDOFS # Pseudo-filesystem framework
options  GEOM_GPT # GUID Partition Tables.
options  COMPAT_43 # Compatible with BSD 4.3 [KEEP THIS!]
options  COMPAT_FREEBSD4 # Compatible with FreeBSD4
options  SCSI_DELAY=15000 # Delay (in ms) before probing SCSI
options  KTRACE # ktrace(1) support
options  SYSVSHM # SYSV-style shared memory
options  SYSVMSG # SYSV-style message queues
options  SYSVSEM # SYSV-style semaphores
options  _KPOSIX_PRIORITY_SCHEDULING # POSIX P1003_1B real-time extensions
options  KBD_INSTALL_CDEV # install a CDEV entry in /dev
options  AHC_REG_PRETTY_PRINT # Print register bitfields in debug
# output.  Adds ~128k to driver.
```

```
options  AHD_REG_PRETTY_PRINT # Print register bitfields in debug
# output.  Adds ~215k to driver.
options  ADAPTIVE_GIANT # Giant mutex is adaptive.

#device

device apic # I/O APIC


# Bus support.  Do not remove isa, even if you have no isa slots
device isa
#device eisa
device pci


# Floppy drives
device fdc


# ATA and ATAPI devices
device ata
device atapicd # ATAPI CDROM drives
#device atapifd # ATAPI floppy drives
options  ATA_STATIC_ID # Static device numbering


# atkbdc0 controls both the keyboard and the PS/2 mouse
device atkbdc # AT keyboard controller
device atkbd # AT keyboard
device psm # PS/2 mouse

device vga # VGA video card driver

device splash # Splash screen and screen saver support


# syscons is the default console driver, resembling an SCO console
device sc


# Enable this for the pcvt (VT220 compatible) console driver
#device vt
#options  XSERVER # support for X server on a vt console
```

```
#options  FAT_CURSOR # start with block cursor

device agp # support several AGP chipsets


# Floating point support - do not disable.
device npx


# Power management support (see NOTES for more options)
#device apm


# Add suspend/resume support for the i8254.
device pmtimer


# PCCARD (PCMCIA) support
# PCMCIA and cardbus bridge support
#device cbb # cardbus (yenta) bridge
#device pccard # PC Card (16-bit) bus
#device cardbus # CardBus (32-bit) bus


# Serial (COM) ports
device sio # 8250, 16[45]50 based serial ports


# Parallel port
device ppc
device ppbus # Parallel port bus (required)
device lpt # Printer
device plip # TCP/IP over parallel
device ppi # Parallel port interface device
#device vpo # Requires scbus and da


# PCI Ethernet NICs.
#device de # DEC/Intel DC21x4x (``Tulip'')
#device em # Intel PRO/1000 adapter Gigabit Ethernet Card
#device ixgb # Intel PRO/10GbE Ethernet Card
#device txp # 3Com 3cR990 (``Typhoon'')
#device vx # 3Com 3c590, 3c595 (``Vortex'')
```

```
# PCI Ethernet NICs that use the common MII bus controller code.
# NOTE: Be sure to keep the 'device miibus' line in order to use these NICs!
device miibus # MII bus support
#device bfe # Broadcom BCM440x 10/100 Ethernet
#device bge # Broadcom BCM570xx Gigabit Ethernet
#device dc # DEC/Intel 21143 and various workalikes
#device fxp # Intel EtherExpress PRO/100B (82557, 82558)
#device lge # Level 1 LXT1001 gigabit ethernet
#device nge # NatSemi DP83820 gigabit ethernet
#device pcn # AMD Am79C97x PCI 10/100 (precedence over 'lnc')
#device re # RealTek 8139C+/8169/8169S/8110S
#device rl # RealTek 8129/8139
#device sf # Adaptec AIC-6915 (``Starfire'')
#device sis # Silicon Integrated Systems SiS 900/SiS 7016
#device sk # SysKonnect SK-984x & SK-982x gigabit Ethernet
#device ste # Sundance ST201 (D-Link DFE-550TX)
#device ti # Alteon Networks Tigon I/II gigabit Ethernet
#device tl # Texas Instruments ThunderLAN
#device tx # SMC EtherPower II (83c170 ``EPIC'')
#device vge # VIA VT612x gigabit ethernet
#device vr # VIA Rhine, Rhine II
#device wb # Winbond W89C840F
device xl # 3Com 3c90x (``Boomerang'', ``Cyclone'')


# ISA Ethernet NICs.  pccard NICs included.
#device cs # Crystal Semiconductor CS89x0 NIC
# 'device ed' requires 'device miibus'
#device ed # NE[12]000, SMC Ultra, 3c503, DS8390 cards
#device ex # Intel EtherExpress Pro/10 and Pro/10+
#device ep # Etherlink III based cards
#device fe # Fujitsu MB8696x based cards
#device ie # EtherExpress 8/16, 3C507, StarLAN 10 etc.
#device lnc # NE2100, NE32-VL Lance Ethernet cards
#device sn # SMC's 9000 series of Ethernet chips
#device xe # Xircom pccard Ethernet


# ISA devices that use the old ISA shims
#device le
```

```
# Pseudo devices.
device loop # Network loopback
device mem # Memory and kernel memory devices
device io # I/O device
device random # Entropy device
device ether # Ethernet support
device sl # Kernel SLIP
device ppp # Kernel PPP
device tun # Packet tunnel.
device pty # Pseudo-ttys (telnet etc)
device md # Memory "disks"
#device gif # IPv6 and IPv4 tunneling
#device faith # IPv6-to-IPv4 relaying (translation)


# The ‘bpf’ device enables the Berkeley Packet Filter.
# Be aware of the administrative consequences of enabling this!
# Note that ’bpf’ is required for DHCP.
device bpf # Berkeley packet filter
```

B.4 The configuration files for the Free BSD Application server setup

B.4.1 /etc/rc.conf

```
nfs_client_enable=''YES''
keymap=''norwegian.iso''
hostname=''Appserver3''
# ifconfig_xl0=''DHCP''
defaultruter=''10.0.0.4''
ifconfig_xl1=''inet 10.0.0.3 netmask 255.255.255.0''
linux_enable=''YES''
```

B.5 The configuratio files for the MPITS

B.5.1 /etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The data network interface
auto eth0
iface eth0 inet static
        address 10.0.0.4
        netmask 255.255.255.0
        network 10.0.0.0
        broadcast 10.0.0.255

# The primary network interface
auto eth1
iface eth1 inet static
        address 129.241.252.120
        netmask 255.255.255.240
        network 129.241.252.112
        broadcast 129.241.252.127
        gateway 129.241.252.113
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 129.241.190.190 129.241.224.51

# The controll data network interface
auto eth2
iface eth2 inet static
        address 129.241.252.124
        netmask 255.255.255.240
        network 129.241.252.112
        broadcast 129.241.252.127
```

B.6 SAMHAIN messages

B.6.1 The initialization message from SAMHAIN

```
From daemon@example.com Sun Nov 26 16:35:11 2006
Return-path: <daemon@example.com> Envelope-to: root@localhost
Delivery-date: Sun, 26 Nov 2006 16:35:11 +0100
Received: from localhost
        ([127.0.0.1] helo=[129.241.252.122] ident=root)
        by filserver2 with smtp (Exim 4.50) id 1GoM2B-0005HT-7f
        for root@localhost; Sun, 26 Nov 2006 16:35:11 +0100
From: <daemon@example.com> To: <root@localhost>
Date: Sun, 26 Nov 2006 16:35:11 CET
Subject: [Samhain at 129.241.252.122] 26-11-2006 16:35:06: CRIT
Message-Id: <E1GoM2B-0005HT-7f@filserver2>

-----BEGIN MESSAGE-----
[2006-11-26T16:35:06+0100] 129.241.252.122
CRIT   :  [2006-11-26T16:35:06+0100]
msg=<POLICY ADDED>,
path=</var/lib/lessdisks/var/www/index.html>,
mode_new=<-rw-r--r-->,
attr_new=<------------>,
imode_new=<33188>,
iattr_new=<0>,
hardlinks_new=<1>,
idevice_new=<0>,
inode_new=<1310807>,
owner_new=<root>,
iowner_new=<0>,
group_new=<root>,
igroup_new=<0>,
size_old=<0>,
  size_new=<56>,
ctime_new=<[2006-11-26T16:34:31]>,
atime_new=<[2006-11-21T20:37:53]>,
mtime_new=<[2006-10-26T18:51:02]>,
chksum_new=<C5B0BC7D2540FD0D1D9F038BF880715136B1FD1EBF20A0A4>

-----BEGIN SIGNATURE-----
16015C5C805CDA823A75D0A732F14F7D2F9637A3CB9C390B
000001 1164555301::129.241.252.122
-----END MESSAGE-----
```

B.6.2 The alert message from SAMHAIN

```
From daemon@example.com Sun Nov 26 16:36:11 2006
Return-path: <daemon@example.com> Envelope-to: root@localhost
Delivery-date: Sun, 26 Nov 2006 16:36:11 +0100
Received: from localhost
        ([127.0.0.1] helo=[129.241.252.122] ident=root)
        by filserver2 with smtp (Exim 4.50) id 1GoM39-0005I7-6j
        for root@localhost; Sun, 26 Nov 2006 16:36:11 +0100
From: <daemon@example.com> To: <root@localhost>
Date: Sun, 26 Nov 2006 16:36:11 CET
Subject: [Samhain at 129.241.252.122] 26-11-2006 16:36:07: CRIT
Message-Id: <E1GoM39-0005I7-6j@filserver2>

-----BEGIN MESSAGE-----
[2006-11-26T16:36:07+0100] 129.241.252.122
CRIT   :  [2006-11-26T16:36:07+0100]
msg=<POLICY [IgnoreNone] --------T->,
path=</var/lib/lessdisks/var/www/index.html>,
mode_old=<-rw-r--r-->,
  mode_new=<-rw-r--r-->,
attr_old=<------------>,
  attr_new=<------------>,
hardlinks_old=<1>, hardlinks_new=<1>,
device_old=<0,0>, device_new=<0,0>,
inode_old=<1310807>, inode_new=<1310807>,
  owner_old=<root>, owner_new=<root>,
iowner_old=<0>, iowner_new=<0>,
group_old=<root>, group_new=<root>,
igroup_old=<0>, igroup_new=<0>,
size_old=<56>, size_new=<56>,
atime_old=<[2006-11-21T20:37:53]>,
atime_new=<[2006-11-26T16:35:34]>,
mtime_old=<[2006-10-26T18:51:02]>,
mtime_new=<[2006-10-26T18:51:02]>,
chksum_old=<C5B0BC7D2540FD0D1D9F038BF880715136B1FD1EBF20A0A4>,
chksum_new=<C5B0BC7D2540FD0D1D9F038BF880715136B1FD1EBF20A0A4>,

-----BEGIN SIGNATURE-----
4972F4A1DA9D57121BDBA3A5629B100B9D54351FD6B02A9D
000002 1164555301::129.241.252.122
-----END MESSAGE-----
```