Global Information Assurance Certification Paper

Securing an OpenBSD 3.5 System for use with Honeyd

GCUX

Practical Assignment

Version 2.1

Option 1

Nicholas J. Smith
August 2004

Table of Contents

1 Abstract

This paper describes the design and implementation of a system that uses a honeypot technology called *Honeyd* to augment the incident response process with respect to fighting worm outbreaks within a company's internal network. Honeyd simulates virtual systems at the network level, which gives it the capability to detect and disable worms.

As there are risks associated with deploying any type of honeypot, the design of the Honeyd server starts with identifying the risks for the environment in which it will be deployed. Risk mitigation steps are formulated to address each risk and these are categorized and numbered in the risk mitigation plan. Although a holistic approach has been taken to mitigating the risks, this paper focuses on the measures that will be directly implemented on the Honeyd server.

The Honeyd server was designed within particular constraints that were identified through feasibility testing prior to the commencement of this pilot project. These real world challenges were compounded by a lack of budget. Although hardware and software compatibility issues guided the choice of server platform, OpenBSD 3.5 was a natural choice because many of the security features that were required to implement the risk mitigation steps are built into the operating system.

Although OpenBSD has a very good default security posture, it was still necessary to take steps to raise the security posture to the level specified in the risk mitigation plan. This baseline posture was further improved by the addition of best practice hardening steps. However, since changes to the configuration of the Honeyd server may need to be made rapidly during an incident, it was necessary to balance security with convenience.

The Honeyd server's role was also given consideration during the design and implementation of the ongoing maintenance procedures. The approach taken was to identify *what* routine tasks need to be performed and then create procedures that define *how* to perform them. Frequently occurring tasks involving lots of steps have been automated with Perl and shell scripts.

The paper concludes with the verification of the operating system configuration in order to demonstrate that all of the server risk mitigation steps have been implemented and fulfill their intended purpose.

2 Document Conventions

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

`Command`	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.
`Filename`	Filenames, paths, and directory names are represented in this style.
`computer output` **`user input`**	The results of a command and other computer output are in this style. User input is shown in bold.
URL	Web URL's are shown in this style.
Quotation	A citation or quotation from a book or web site is in this style.

It should be noted that some of the command output has been reformatted in order to make it more readable in this document format.

3 Introduction

3.1 Overview

The purpose of this paper is to describe the design and implementation of a server that will be deployed as part of a pilot project to determine the effectiveness of using honeypot technology to detect and disable worms within an organization's intranet.

With the number and effectiveness of worm outbreaks on the rise and the time between a vulnerability announcement and the release of an exploit on the decline, there is an increasing need to develop technology to help defend against the growing malware threat. Several approaches to tackling the problem are being pursued in the commercial and open source arenas. Of these approaches, the work of Oudot[1] which uses a honeypot technology called Honeyd[2] is of interest to us due to its flexibility, simplicity and price (free, as in beer).

Since this type of technology is seen to be bleeding edge, a great deal of care needs to be taken when deploying it on a production network. For this reason, a disciplined approach is necessary in the design, implementation and operational phases of the project.

3.2 Honeypot Technology

Honeypots have been around for a number of years in one form or another. However, it is only in more recent times that their profile has been raised, largely thanks to the efforts of many talented individuals who volunteer their time to community projects such as The Honeynet Project[3].

A succinct and flexible definition of a honeypot was created by members of the SecurityFocus Honeypots mailing list[4] through a consensus process that was facilitated by Lance Spitzner:

> *An information system resource whose value lies in unauthorized or illicit use of that resource*[5].

Part of the value proposition of a honeypot is that since it is an unadvertised, non-production system, all attempts to use it are suspect and, therefore, worthy of investigation.

[1] Fighting Internet Worms with Honeypots.
[2] http://www.citi.umich.edu/u/provos/honeyd/
[3] http://project.honeynet.org/
[4] http://www.securityfocus.com/archive/119
[5] http://www.securityfocus.com/archive/119/322363/2003-05-22/2003-05-28/0

The three main functions of a honeypot are:

1. Collection. Can provide us with small sets of high-value data.

2. Detection. Can provide us with early warnings of attacks with fewer false positives.

3. Diversion. Can provide us with more time to fix vulnerabilities by diverting attacks away from real hosts.

For the purpose of this project we are going to focus on the detection capabilities of honeypots.

3.3 Honeyd

There are at least three issues with honeypots that need to be addressed in order to make the approach effective against worm attacks:

1. A traditional honeypot only sees the traffic that is directed at it. In a class A network (e.g. 10.0.0.0), a single honeypot would be ineffective because the chance of it being found by a worm that is randomly scanning the network is extremely low. In order to increase our chances, we would need to deploy more honeypots.

2. Honeypots can be difficult and expensive to set-up and run. If we multiply the effort and expense by the number of honeypots needed to be effective, we quickly find that the approach becomes impractical and prohibitively expensive.

3. Deploying a honeypot can be risky.

The issue of risk will be dealt with more thoroughly in the next section. In general, though, high-interaction honeypots carry more risk than low-interaction ones because they are more complex to deploy and maintain.

The first two issues can be addressed by a relatively new honeypot technology called *Honeyd*. In the words of its creator, Niels Provos, Honeyd is "*a framework for virtual honeypots that simulates virtual computer systems at the network level*". The key here is the virtualization of honeypots, which enables us to deploy large numbers of honeypots with relative ease.

In terms of our goal of detecting worm activity, Honeyd allows us to monitor thousands of addresses by populating unused IP address space with virtual systems. These systems can be configured with the appropriate network services that make them appear vulnerable to a worm.

In reality, these virtual systems will reside within our Honeyd server, the design and implementation of which is the real focus of this paper.

- END OF SECTION -

4 Server Specification and Risk Mitigation Plan

4.1 Overview

The risks associated with deploying the Honeyd server need to be considered before the start of the design process because some of the risk mitigation steps may need to be included in the server specification. For example, we might try to eliminate the need to offer a remote session service (and thereby reduce risk) by choosing hardware that provides a means of securely accessing the system console independently of the operating system. Other risk mitigation steps may take place during implementation. For example, we might need to disable unnecessary network services that are enabled by default. Based on this premise, the approach taken is as follows:

1. Provide a high level statement of the server's role.
2. Identify the risks associated with deploying this type of server along with the mitigation step(s) for each risk.
3. Create the risk mitigation plan by rationalizing and categorizing the risk mitigation steps identified in the previous step.
4. Choose the server platform.
5. Choose the versions of the third party software.
6. Create the server specification.
7. Design the filesystem layout.

In broad terms, steps 1 to 3 focus on *what* needs to be done and steps 4 to 7 focus on *how* it will be done.

4.2 Server Role

The main purpose of the Honeyd server is to augment parts of the existing incident response process with respect to handling worm attacks:

> Identification. Detect potential worm activity by deploying virtual honeypots in order to instrument unused IP address space within the organization's network. Once worm activity has been detected, identification is possible by determining its exploit mechanism and capturing its payload.

Containment. Slow down or stop the spread of a worm. With a total of over 16 million IP addresses available in a class A network such as 10.0.0.0, there is plenty of scope for increasing the chance that a worm connects to a virtual honeypot rather than a real system. If the worm connects to a virtual system it is wasting its time and will, therefore, be slowed to some extent. Adding Honeyd's ability to act as a tar pit, we can further slow the worm's progress. Data from Honeyd can also be fed to the incident response team to further assist with containment.

Eradication. Under certain circumstances it may be possible for the incident response team to configure countermeasures into Honeyd to enable it to neutralize worms on infected systems.

In time we may wish to either expand or change the role of Honeyd, however, the sole purpose of the server will always be to run Honeyd.

4.3 Risks

Regardless of our choice of server platform, there are risks associated with deploying honeypot technology on a production network. In general, the risks associated with deploying a virtual honeypot such as Honeyd are lower than deploying a honeypot based on a real system. The risk is further decreased by the fact that the Honeyd server will be deployed on an intranet and will not be exposed to the Internet.

Although the intention is to expose Honeyd to worms, there is a chance that there could be exposure to entities that are intent on attacking the server hosting Honeyd. Since these entities are likely to be smarter than worms, their actions pose the greatest risk.

The following table identifies the risks of running the Honeyd server along with the mitigation steps.

Risk	Comment	How it happens	Risk mitigation
The deployment of Honeyd is deemed to be misuse of the organization's information resources.	A honeypot is a tool that is used to enforce security policy. We need to ensure that policy clearly defines how the honeypot can be used within the organization.	Various ways, including lack of or bad communication, intra-company politics, etc.	Create clear policy for the use of Honeyd.
			Communicate the policy to the appropriate parties.
			Get C-level buy-in for the deployment.

(cont ...)

Unauthorized access to the server.	The risk associated with unauthorized access is related to the consequences, i.e. what the unauthorized user does after gaining access. Although each consequence is identified as a separate risk in this table, it is important to identify the issue of unauthorized access in order to address mitigation techniques.	Exploit of known OS vulnerability.	Apply the latest OS patches.
		Compromise of Honeyd daemon.	Run a version of Honeyd with no known vulnerabilities.
			Run Honeyd in a sandbox or jail environment.
			Implement a stack protection mechanism to help prevent buffer overflow attacks.
		Compromise of services running on the server hosting Honeyd.	Run the minimum number of services on the server.
			Use IP-based access control to limit access to services.
			Run services in a sandbox or jail environment, where possible.
			Implement a stack protection mechanism to help prevent buffer overflow attacks.
			Run NIDS on a separate server to monitor connections to the Honeyd server.
		Physical access to the system.	Locate the Honeyd server in a data centre where preventative and detective physical access control issues have been addressed.
		Brute-force password attack.	Monitor log files to detect unauthorized access attempts.
		Password sniffing or session hijacking.	Eliminate the need for remote sessions or used encrypted sessions.

(cont…)

Honeyd server causes disruption of services on the production network.	Honeyd is a powerful tool that could easily impact production services on the network if used incorrectly. This might range from Denial of Service through to redirection of connections to systems of an attacker's choosing. Included in this risk is rendering Honeyd useless in order to allow a worm attack to succeed.	Misconfiguration of Honeyd by authorized users.	Ensure that all new configurations undergo peer review.
			Ensure that the change management process is followed when introducing a new configuration into the production environment.
		An unauthorized user configures Honeyd to disrupt services running on the network or render Honeyd useless.	Design and implement the file system in a manner that will make it difficult to make unauthorized system changes.
			Monitor log files to detect unauthorized activity on the system.
			Run file integrity checking software to detect changes to essential files.
Server is used as a launch pad to attack other systems.	This is usually the number one risk associated with the deployment of honeypots. Since the Honeyd server will be deployed on an intranet, we do not have to concern ourselves with it being used to directly attack systems belonging to other people. However, we do need to concern ourselves with attacks against our own systems.	Once an attacker has compromised a system, s/he will typically download a toolkit that will enable them to cover tracks, scan the network for vulnerabilities and launch attacks against other systems.	Run a packet filter on a separate system to control outbound connections from the Honeyd server.
			Run NIDS on a separate server to monitor connections from the Honeyd server.
			Send logging data to a centralized syslog server to make it harder for the attacker to cover tracks.
			Design and implement the file system in a manner that will make it difficult to make unauthorized system changes.
			Run file integrity checking software to detect changes to system binaries to make it more difficult for the attacker to install a rootkit.

(cont…)

Server suffers catastrophic failure.	Things happen, so we need to be prepared.	Hardware failure.	Keep spare hardware – both parts and complete systems.
			Purchase a hardware support contract for the system.
		Logical software error or a malicious act.	Implement a backup and restore process.
The server becomes unavailable.	This is not a mission critical system, but nevertheless it will potentially serve an important role in the organization's fight against malware.	Power failure, network outage, etc.	Locate the Honeyd server in the data centre, where these types of facilities issues have been addressed.
Alerts from the Honeyd server and associated systems, go unnoticed.		Lack of time to monitor activity.	Send alerts to the Operations group because they are dedicated to monitoring and provide 24x7 coverage.
		Failure of a transport mechanism.	Send regular heartbeat messages to Operations to verify the transport mechanisms.

It should be noted that in discussing the consequences of a server compromise, we are assuming that an attacker has been able to defeat some of the measures that we have put in place to prevent such a compromise. The point is that, from time to time, some of our defenses will fail. Therefore, it is important to apply the principle of defense in depth so that other layers of our security architecture will prevent an attack from progressing any further. For example, although Honeyd will be run in a sandbox or jail, we cannot assume that an attacker will not be able to break out of it. After a successful jail break, the attacker might attempt to download files to the system, in which case our separate packet filtering system should detect, log and possibly respond to the activity, and our log monitoring software should send an alert.

The Honeyd server is not a mission critical system. However, because of its intended role is has the potential to disrupt other systems, which themselves may be mission critical. In addition, the project has high profile within certain influential parts of the organization so it is essential that problems do not occur as a result of the deployment of the server.

4.4 Risk Mitigation Plan

The risk mitigation steps identified in the above table are summarized in the table below. Duplicates have been removed and each one has been numbered for ease of reference.

The risk mitigation steps have been categorized as follows.

- **Policy**. This relates to measures that need to be included in the organization's security policy.
- **Management**. This relates to measures that need to be addressed by management.
- **Server**. This relates to measures that will be implemented on the Honeyd server.
- **Architecture**. This relates to measures that will be implemented on separate systems within the environment.

Details relating to policy, management and architecture are beyond the scope of this document. However, it is worth mentioning that the architectural risk mitigation steps that have been identified here are considered to be good practice when deploying a honeypot of any type.

Reference Number	Risk Mitigation Step (RMS)	Type
1	Create clear policy for the use of Honeyd.	Policy
2	Communicate the policy to the appropriate parties.	Management
3	Get C-level buy-in for the deployment.	Management
4	Apply the latest OS patches.	Server
5	Run a version of Honeyd with no known vulnerabilities.	Server
6	Run Honeyd in a sandbox or jail environment.	Server
7	Implement a stack protection mechanism to help prevent buffer overflow attacks.	Server
8	Run the minimum number of services on the server.	Server
9	Use IP-based access control to limit access to services.	Server
10	Run services in a sandbox or jail environment, where possible.	Server
11	Run NIDS on a separate server to monitor connections to and from the Honeyd server.	Architecture
12	Locate the Honeyd server in a data centre.	Architecture
13	Monitor log files to detect unauthorized access attempts (assumes that log files are sent to a centralized syslog server).	Policy
14	Eliminate the need for remote sessions or use encrypted sessions.	Server

(cont…)

15	Ensure that all new configurations undergo peer review.	Policy
16	Ensure that the change management process is followed when introducing a new configuration into the production environment.	Policy
17	Design and implement the file system in a manner that will make it difficult to make unauthorized system changes.	Server
18	Run file integrity checking software to detect changes to essential files such as configuration files and system binaries.	Server
19	Run a packet filter on a separate system to control outbound connections from the Honeyd server.	Architecture
20	Send logging data to a centralized syslog server to make it harder for the attacker to cover tracks.	Server
21	Keep spare hardware (parts and complete systems).	Management
22	Purchase a hardware support contract for the system.	Management
23	Implement a backup and restore process.	Server
24	Send alerts to the Operations group.	Server
25	Send regular heartbeats.	Server

4.5 Platform Choice

Platform choice is about selecting a combination of hardware and operating system that:

- Will enable us to meet security requirements.
- Will enable us to meet performance / capacity requirements.
- Is freely available (again as in beer because there is no budget for this pilot project).
- Is familiar to us – this is a risky project (in many senses) so it is better that we know what we are doing.

Our choice of hardware boils down to SPARC and Intel architectures. Since this is a low / no budget production, our only options are:

- Sun Enterprise 250 (E250).
- Compaq Proliant 5500.

In terms of OS choice, the usual suspects are:

- Solaris.
- Red Hat Linux.
- OpenBSD.

From a security standpoint, the hardware preference is the Sun E250 because it incorporates a Remote Services Control (RSC) card that enables us to connect to the system independently the OS. The RSC card is a lights out management feature that enables a remote administrator to interact with the system as though s/he were in front of the console and includes the ability to power the server on and off. When combined with a console server that supports SSH, the RSC card provides a secure means of managing the server. This feature could allow us to address RMS #8 by eliminating the need to run a remote session service. In addition, the E250 is a headless system, which would be accommodated in the data centre more readily. This would make is easier for us to address RMS #12.

Although the capacity requirements of Honeyd are unknown at this stage, from a performance standpoint, the hardware preference is the Compaq Proliant 5500 because it has quad Pentium III Xeon 500 MHz processors and 1GB of RAM, making it the more powerful machine of the two. Having said that, a nice feature of the Honeyd framework is that it can be run in a distributed fashion, which would enable the load to be shared by more than one machine.

Taking security and performance into consideration, the E250 is the hardware platform of choice. The RSC card gives it the edge over the Compaq in terms of meeting security requirements and although not as powerful as the Compaq machine, we do have more than one spare E250. Building and maintaining more than one system requires more effort, but it also means that not all of our eggs are in one basket.

Based on this hardware preference, we have to eliminate one of our OS options; up until version 6.2, Red Hat would run on both SPARC and Intel hardware, however, this is no longer the case. Since RH 6.2 is somewhat out of date, it will not be considered for the Honeyd server.

Although there are many other Linux distributions that run on SPARC, there would be a learning curve involved in using it. The lack of familiarity with the product could lead to security issues being missed so it is felt that it is better to stick with what we know.

At this point it is prudent to check that there are no hardware compatibility issues with the two remaining OS candidates. We know that there are no issues with running Solaris on the E250 and the OpenBSD web site[6] confirms that there should be no problems with running OpenBSD on the E250.

Some people might be tempted to select Solaris because it seems more fitting to run Sun software on Sun hardware. However, Solaris takes a lot more work to secure than OpenBSD because of the latter's "Secure by default" configuration. In addition, OpenBSD has some built-in features such as integrated crypto, W^X

[6] http://www.openbsd.org/plat.html

and ProPolice, which would make it that much easier to implement the required risk mitigation measures.

4.6 Reality Check

Before a decision is made about the OS, we need to determine whether the Honeyd software will run on the candidate operating systems.

According to the Honeyd website, the software should run on Linux, *BSD and Solaris. However, my attempts to build Honeyd on a Solaris system in the lab failed because I could not get the software to compile. Searching the SecurityFocus honeypot archives revealed that other people had also experienced problems with Honeyd on Solaris, but unfortunately no one had come up with a fix. Honeyd's creator, Niels Provos, was very helpful in fixing some of the issues that I had, but as he did not have access to a Solaris system he was only able to go so far.

Based on this discovery, I decided to drop Solaris from the list of OS candidates and thus arrived at a combination of OpenBSD on SPARC. This brings us to the second reality check.

After purchasing the latest release of OpenBSD (3.5 at the time of writing), I proceeded with installing it on the Sun E250. Everything seemed to go fine with the installation, but the system consistently froze midway through the boot sequence. The OpenBSD website and *bugs* mailing list had nothing to confirm that this was a known issue so I sent in a bug report. Someone replied to me off list and indicated that he had experienced a similar problem and was waiting for a patch from the OpenBSD development team. In a later note he said that the patch had not helped. Since I did not get a response from anyone in the OpenBSD development team, I decided to switch to Plan B.

Plan B was to switch to the Compaq Proliant machine. Although is does not have the equivalent of an RSC card, we can run OpenSSH for remote access and still comply with the security requirements of the system (RMS #8 states that the minimum number of services should be run and RMS #14 states that encrypted sessions should be used if the need for remote sessions cannot be eliminated). It was decided to stay with OpenBSD 3.5 because, ultimately, when the bugs are ironed out of the current SPARC version, I plan to use the E250 as the hardware platform.

4.7 Third Party Software Version Choice

Given the problems that had been experienced with Honeyd on Solaris, it was thought prudent to check that it would run on OpenBSD.

A nice feature of *BSD systems is the ports tree[7], which makes it very easy to install some third party software. Honeyd is included in the OpenBSD 3.5 ports tree, but unfortunately it is version 0.7a, which contains a remote detection vulnerability[8]. As this would contravene RMS #5 and is generally not a good idea anyway, it was necessary to choose the latest version (0.8b at the time of writing).

According to the Honeyd documentation in the ports tree, Honeyd is dependent on libtool 1.3.5 and libdnet 1.7. After building both of these packages, I was able to verify the build and operation of Honeyd 0.8b.

Although there is no dependency from a build point of view, Honeyd needs arpd[9] to operate. The purpose of arpd is to listen to ARP requests and respond to the ones that correspond to the IP addresses that we are simulating with Honeyd. The ports tree contains arpd 0.1, which is dependent on autoconf 2.13, libdnet 1.7 and metaauto 0.1.

Although arpd does not contain any known vulnerabilities, the current version (0.2 at the time of writing) has some new features, which include the ability to specify multiple IP address / networks on the command line. As this is likely to make things easier for us, it was decided that version 0.2 would be used. This version has a dependency on a version of libevent that is more recent than the one that is included in the OpenBSD 3.5 base install. At the time of writing, libevent 0.8a is the most recent version and, therefore, the one that will be used to build arpd 0.2.

4.8 Server Specification

Hardware	Compaq Proliant 5500: • Quad Pentium III Xeon 500 MHz processors • 1 GB RAM • Smart Array 3200 Controller • 4 x 9.1 GB Ultra2 SCSI 10K drives (logically configured as one 26 GB RAID 5 drive). • Intel 10/100 NIC
Operating System	OpenBSD 3.5 (i386)

[7] http://www.openbsd.org/ports.html
[8] http://www.securityfocus.com/bid/9464
[9] http://www.honeyd.org/tools.php

(cont…)

Risk Mitigations Steps directly addressed by built-in OS features	• Sandboxing with *systrace* (RMS #6) • Stack protection with *W^X and ProPolice* (RMS #7) • IP-based access control with *pf* (RMS #9) • Jailing with *privilege separation* (RMS #10) • Encrypted sessions with *OpenSSH* (RMS #14) • File system protection with *immutable files* and *securelevel* (RMS #17) • File integrity checking with *mtree* (RMS #18)
Third Party Software	Binary installed from OpenBSD 3.5 CDROM packages: • bash 2.05b Built from OpenBSD 3.5 ports tree: • autoconf 2.13 • autoconf 2.52 • libdnet 1.7 • libtool 1.3.5 • metaauto 0.1 • ntp 4.1.1c Built from source independently of ports tree: • arpd 0.2 • honeyd 0.8b • libevent 0.8a
Additional processes running on the system	• arpd • honeyd • ntpd • pflogd
Network services offered by the host	• ssh (tcp port 22) Note: Although network services will be simulated by Honeyd on various udp and tcp ports, they are not considered as network services offered by the host.
Authorized system users	During the pilot phase of this project only I will have access to this system. Naturally, I will have normal user and root access. Direct root login will only be allowed at the console however such practice is reserved for emergency use only. For normal use I will log in using my user account and use sudo to execute commands that require root privilege.

4.9 Design of the Filesystem Layout

The OpenBSD development team has gone to great lengths to ensure that programs and files are kept in the correct place. Since the placement and grouping of files is very logical, it makes the job of designing the filesystem layout that much easier.

An important point to bear in mind is that the /bin and /sbin directories should reside on the / partition so that the system is able to boot.

For security and maintenance purposes, the approach that will be taken here is to create separate filesystems for /, /usr, /usr/ports, /usr/src, /usr/local, /home, /var, and /tmp. As the disk is large, we can afford to be generous with the sizing of the partitions.

Filesystem	Size (MB)	Partition	Default Mount Option(s)	Final Mount Option(s)	Comment
/usr	3072	d	rw async local nodev	rw async local nodev	Contains files belonging to the base operating system.
/usr/ports	3072	e	rw async local nodev nosuid	rw async local nodev nosuid	Contains the ports tree. By making it a separate filesystem, the ports tree can be quickly deleted by creating a new file system on /usr/ports. In the final configuration the entire directory tree will be made immutable to prevent an attacker from using it to build programs.
/usr/src	3072	f	rw async local nodev nosuid	rw async local nodev nosuid	Contains the system source. By making it a separate filesystem, the source tree can be quickly deleted by creating a new file system on /usr/src. In the final configuration the entire directory tree will be made immutable to prevent an attacker from using it to build programs.
/usr/local	1024	g	rw async local nodev	rw async local nodev	Third party software or files that are not part of the base installation should be installed here.

(cont…)

/home	512	h	rw async local nodev nosuid	rw async local nodev **noexec** nosuid	Contains files belonging to users. Although this is not intended to be a multi-user system, it is good practice to make it a separate filesystem in order to prevent the / filesystem from filling up and causing a system crash. In the final configuration it will be mounted noexec to restrict malicious actions that could result from a compromised user account.
/var	12459	j	rw async local nodev nosuid	rw async local nodev nosuid	Contains variable data such as log files. It is good practice to make it a separate filesystem in order to prevent the / filesystem from filling up and causing a system crash.
/tmp	512	i	rw async local nodev nosuid	rw async local nodev nosuid	Contains temporary files. Again it is good practice to make it a separate filesystem in order to prevent the / filesystem from filling up and causing a system crash.
/	256	a	rw async local	rw async local	Contains everything else.
Swap	2048	b			2 x the size of physical memory.

- END OF SECTION -

5 Steps to Install and Harden the Server

5.1 Overview

Despite OpenBSD's "secure by default" approach, there are still steps that can be taken to improve the security posture of a system. In general, the measures that are put in place will depend on:

1. The risks that *need* to be mitigated, i.e. implement the minimum or baseline security posture based on the risk mitigation steps that were derived from the risk analysis.

2. The effort that we are willing to expend to perform extra "best practice" steps that take the security posture of the system beyond the baseline requirement.

3. The tradeoffs that we are willing to accept in terms of security versus convenience.

For the Honeyd server we will add a few high-value best practice steps that can be implemented with relatively little effort in order to bolster the baseline security posture. Although it would be possible to perform many best practice steps, it is necessary to balance these measures against the necessity to streamline maintenance processes so that the Honeyd server is effective in its role of augmenting the incident response process, i.e. the system should not be cumbersome to use or maintain.

The server installation and hardening steps address the following issues.

- **Preparation**. Gathering and reviewing of information required for the installation of the operating system.

- **OS installation**. Installing OpenBSD in 5 easy steps.

- **Afterboot**. Miscellaneous system administration and security enhancement tasks to perform after the OS installation.

- **OS patches**. Obtaining, applying and verifying operating system patches.

- **Services and processes**. Eliminating unnecessary services.

- **Logging**. Configuring syslogd to log to a remote server.

- **Warning banners**. Setting up warning messages that are displayed before and after login.
- **OpenSSH**. Enhancing the default configuration of OpenSSH for encrypted remote sessions.
- **NTP**. Installing and configuring NTP for synchronizing the system clock.
- **pf**. Enabling and configuring OpenBSD's built-in packet filtering software for IP-based access control.
- **Honeyd**. Installing, configuring, sandboxing and verifying the Honeyd software.
- **System Heartbeat**. Configuring a heartbeat to test the operation of alerting mechanisms.
- **File integrity checker**. Configuring mtree to check for changes to key files.
- **Filesystem access control**. Setting the system immutable flag on important files and setting the security level.

Approximate times are provided for performing tasks associated with each phase. However, it is possible to reduce the overall time required by delaying some tasks. For example, after applying some of the OS patches a kernel rebuild is required. However, a kernel rebuild is also required to disable IPv6. By eliminating the need to perform two kernel rebuilds, at least 30 minutes can be saved on the overall time that it takes to install and harden the server.

The overall time taken to perform the install and hardening steps described in this section was approximately 5 hours. It should be noted that this time is somewhat dependent on the hardware.

5.2 Preparation

Objective: Gather and review information required for installation of the operating system.

There are a couple of things to keep in mind when preparing to install an operating system:

1. Normally, it is good practice to review the hardware components in the system in case there are compatibility or driver issues. However, since problems were identified during the design and specification stage (see

section 4.6, “Reality Check”) it is already known that the Compaq Proliant 5500 system works fine with OpenBSD 3.5 because it has been tested.

2. The server could be compromised before it is hardened. The HoneyNet project has observed servers being compromised within minutes of being connected to a network. Although these servers were honeypots, they do have something in common with newly installed servers: they often use the default operating system configuration. For this reason, it is good practice to install the OS on an isolated network segment. Since some of the risk mitigation steps for running the Honeyd server include setting up separate architectural components, we can leverage the use of the packet filtering device to provide us with an isolated network for our installation. The installation environment can be configured to emulate the final production environment in which the Honeyd server will be deployed so it will not be necessary to make any last minute configuration changes such as IP settings. The entire installation and hardening process will be performed behind locked doors in the lab.

Preparation steps specific to the Compaq system include deleting the old system configuration and using the Compaq SmartStart CD to reconfigure the system as required. This included setting the time and date to the current UTC time and disabling the ‘boot from floppy’ feature.

Finally, it is useful to have the network configuration information available to speed up the installation process. The network configuration information for the Honeyd server is shown in the following table.

Hostname	matrix
IP Address	10.192.168.10
Netmask	255.255.255.0
Domain	fake.world
DNS Server(s)	10.192.169.53 10.160.169.53 10.128.169.53
Default Gateway	10.192.168.1
Loghost	10.192.170.51 10.160.170.51 10.128.170.51
NTP Server(s)	10.192.171.123 10.160.171.123 10.128.171.123
SMTP Server	10.192.172.25

5.3 OS Installation in 5 Easy Steps

Objective: Install the minimal software components required to enable the system to fulfill the role set out for the Honeyd server.

Time required: 15 minutes.

The operating system will be installed from CD ROM. Not only is this more convenient, it is slightly more secure than the network-based methods (FTP, NFS and HTTP) since it eliminates the potential for a man in the middle attack. The assumption being made here is that the software on the CD is trusted and has not been the subject of a man in the middle attack of the physical kind.

Since installing OpenBSD is quick and easy, the few steps that are required are shown here for completeness.

5.3.1 Step 1: Boot from the CD

The installation program prompts with the options available. Since this is a new installation, the (I)nstall option is chosen.

```
erase ^?, werase ^W, kill ^U, intr ^C, status ^T
(I)nstall, (U)pgrade or (S)hell? I
```

Some preamble follows and the program prompts for answers to some basic set-up questions. There is nothing security related here.

```
Welcome to the OpenBSD/i386 3.5 install program.

This program will help you install OpenBSD in a simple and rational
way. At any prompt except password prompts you can run a shell command
by typing '!foo', or escape to a shell by typing '!'. Default answers
are shown in []'s and are selected by pressing RETURN. At any time you
can exit this program by pressing Control-C and then RETURN, but
quitting during an install can leave your system in an inconsistent
state.

Terminal type? [vt220] enter
Do you wish to select a keyboard encoding table? [no] enter

IS YOUR DATA BACKED UP? As with anything that modifies disk contents,
this program can cause SIGNIFICANT data loss.

It is often helpful to have the installation notes handy. For complex
disk configurations, relevant disk hardware manuals and a calculator
are useful.

Proceed with install? [no] yes
```

```
Cool!  Let's get to it...
```

5.3.2 Step 2: Partition the Filesystem

After prompting for the disk that will be initialized, the installation program calls `disklabel` in edit mode to enable the disk to be partitioned:

```
You will now initialize the disk(s) that OpenBSD will use. To enable
all available security features you should configure the disk(s) to
allow the creation of separate filesystems for /, /tmp, /var, /usr, and
/home.

Available disks are: sd0.
Which one is the root disk? (or 'done') [done] sd0
This platform requires that partition offsets/sizes be on cylinder
boundaries.
Partition offsets/sizes will be rounded to the nearest cylinder
automatically.
```

Since there is only one logical drive there are no special considerations to make in terms of configuration. The filesystem layout was determined during the design and specification phase, so it is a simple matter of using disklabel to set up the disk partitions accordingly. In the interest of brevity, only the resulting partition table is shown here.

```
16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:   522208       32    4.2BSD     2048 16384    16   # /
  b:  4194240   522240      swap                        #
  c: 53309280        0    unused        0     0        #
  d:  6291360  4716480    4.2BSD     2048 16384    16   # /usr
  e:  6291360 11007840    4.2BSD     2048 16384    16   # /usr/ports
  f:  6291360 17299200    4.2BSD     2048 16384    16   # /usr/src
  g:  2097120 23590560    4.2BSD     2048 16384    16   # /usr/local
  h:  1052640 25687680    4.2BSD     2048 16384    16   # /home
  i:  1052640 26740320    4.2BSD     2048 16384    16   # /tmp
  j: 25516320 27792960    4.2BSD     2048 16384    16   # /var
```

Once the disk has been partitioned, the installation program prompts for confirmation to proceed with the formatting of the partitions:

```
The next step creates a filesystem on each partition, ERASING existing
data.

Are you really sure that you're ready to proceed? [no] yes
```

5.3.3 Step 3: Configure Networking

Networking configuration is very straightforward; just respond to the prompts from the installation program:

```
System hostname? (short form, e.g. 'foo') matrix
Configure the network? [yes] yes
Available interfaces are: fxp0.
Which one do you wish to initialize? (or 'done') [fxp0] enter
Symbolic (host) name for fxp0? [matrix] enter
The default media for fxp0 is
        media: Ethernet autoselect (10BaseT)
Do you want to change the default media? [no] enter
IP address for fxp0? (or 'dhcp') 10.192.168.10
Netmask? [255.255.255.0] 255.255.255.0
No more interfaces to initialize.
DNS domain name? (e.g. 'bar.com') [my.domain] fake.world
DNS nameserver? (IP address or 'none') [none] 10.192.169.53
Use the nameserver now? [yes] enter
Default route? (IP address, 'dhcp' or 'none') 10.192.168.1
add net default: gateway 10.192.168.1
Edit hosts with ed? [no] enter
Do you want to do any manual network configuration? [no] enter
```

At the end of the network configuration section the installation program will prompt for the root password to be entered.

5.3.4 Step 4: Install the Base Software Set

The OpenBSD development team has worked hard to logically group the system files and programs into installation sets. As shown in the following screen output from the installation program, there are 12 installation sets.

```
You will now specify the location and names of the install sets you
want to load. You will be able to repeat this step until all of your
sets have been successfully loaded. If you are not sure what sets to
install, refer to the installation notes for details on the contents of
each.

Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or
(t)ape device; or a (f)tp, (n)fs or (h)ttp server.
Where are the install sets? (or 'done') c
Available CD-ROMs are: cd0.
Which one contains the install media? (or 'done') [cd0] enter

Pathname to the sets? (or 'done') [3.5/i386] enter

The following sets are available. Enter a filename, 'all' to select
all the sets, or 'done'. You may de-select a set by prepending a '-'
to its name.
```

```
[X] bsd
[ ] bsd.rd
[X] base35.tgz
[X] etc35.tgz
[X] misc35.tgz
[X] comp35.tgz
[X] man35.tgz
[X] game35.tgz
[ ] xbase35.tgz
[ ] xshare35.tgz
[ ] xfont35.tgz
[ ] xserv35.tgz
```

The objective is to specify the smallest functional OS image. At a minimum, the following sets must be installed in order for the system to function correctly:

- **bsd**. The BSD kernel.
- **base35.tgz**. The base set contains the core system functionality.
- **etc35.tgz**. This set contains the system configuration files.

The `comp35.tgz` file contains C, C++ and F77 compilers. Normally, it would not be good practice to install such software on this type of server because it could be of use to an attacker who manages to gain access to the system. From a convenience point of view, omitting this set would make installing third party software more difficult, but not impossible. If this was the only consideration, we would choose security over convenience and not install this software set.

Unfortunately, the compiler software is required for patching the system because OpenBSD patches are not distributed in binary format. They are supplied as diff files that must be applied to the OpenBSD source code, which must then be compiled. Given that security is the priority, we need to install the compiler software in order to allow us to be able to apply patches that fix security issues.

A further wrinkle in the issue of installing the compiler software is that it is not installed as a package (i.e. with the pkg_add command) which means there is no supported way of uninstalling it after we have finished using it.

The bottom line is that the compiler software will have to be installed. However, the risk of it being used against us will be mitigated by the other layers of our defense-in-depth approach.

The `man35.tgz` file contains the manual pages and other online information. The OpenBSD documentation recommends that this set is installed. In practice, it has been found that the files from this set need to be present in order for the patching process to work cleanly. Although the installation of this set can be construed as

being a convenience, it is not to the detriment of the security of this system because all of the OpenBSD manual pages are available from the OpenBSD web site[10].

The X Window system (`x*35.tgz`) will not be installed on this system as it could introduce security issues. The games (`game35.tgz`) set is not required and will also be omitted.

Continuing with the installation, we specify that only the `bsd35.tgz`, `base35.tgz`, `etc35.tgz`, `comp35.tgz` and `man35.tgz` sets are installed. When we are done, the system proceeds with the installation of these sets:

```
        [X] bsd
        [ ] bsd.rd
        [X] base35.tgz
        [X] etc35.tgz
        [ ] misc35.tgz
        [X] comp35.tgz
        [X] man35.tgz
        [ ] game35.tgz
        [ ] xbase35.tgz
        [ ] xshare35.tgz
        [ ] xfont35.tgz
        [ ] xserv35.tgz

File name? (or 'done') [bsd.rd] done
Ready to install sets? [yes] yes
Getting bsd ...
100% |***************************************************|  4956 KB
00:04
Getting base35.tgz ...
100% |***************************************************| 30270 KB
01:15
Getting etc35.tgz ...
100% |***************************************************|  1603 KB
00:06
Getting comp35.tgz ...
100% |***************************************************| 17358 KB
00:48
Getting man35.tgz ...
100% |***************************************************|  6516 KB
00:25
```

5.3.5 Step 5: Post installation

Once the software sets have been installed, the installation program prompts for some final configuration information.

[10] http://www.openbsd.org/cgi-bin/man.cgi

The choice of time zone is important in our environment because the network spans the globe. All of the systems that are connected to the network have their system clocks set to UTC (zero offset from GMT) in order to make it easier to correlate log entries in the event of a security incident.

```
Do you wish sshd(8) to be started by default? [yes] enter
Do you expect to run the X Window System? [yes] no
Saving configuration files...done.
Generating initial host.random file...done.
What timezone are you in? ('?' for list) [Canada/Mountain] GMT
Setting local timezone to 'GMT'...done.
Making all device nodes...done.
Installing boot block...
Boot: /mnt/boot
proto: /usr/mdec/biosboot
device: /dev/rsd0c
/usr/mdec/biosboot: entry point 0
proto bootblock size 512
/mnt/boot is 3 blocks x 16384 bytes
fs block shift 2: part offset 32; inode block 24, offset 11688
using MBR partition 3: type 166 (0xa6) offset 32 (0x20)
done.
```

When the boot block has been installed, the following text will be displayed. Type `halt` to reboot the system and then remove the CD.

```
CONGRATULATIONS! Your OpenBSD install has been successfully completed!
To boot the new system, enter halt at the command prompt. Once the
system has halted, reset the machine and boot from the disk.
# halt
```

5.4 Afterboot

Objective: Perform miscellaneous system administration and best practice security enhancement tasks.

Time required: 15 minutes.

If we had installed the man pages on the Honeyd server, typing `man afterboot`[11] would display a manual page describing some best practice tasks that the system administrator should perform after the first system boot. There are approximately 30 tasks in the afterboot 'to do' list, which can be roughly divided into:

- System admin tasks.

[11] http://www.openbsd.org/cgi-bin/man.cgi?query=afterboot

- System checks.
- Security enhancements.

Not all of the tasks are applicable to the Honeyd server and some of the security-related tasks will be covered later in our own hardening steps. However, there are some tasks that should be performed now. These include:

- Creating a user account and specifying group membership.
- Setting up the `/etc/sudoers` file.
- Adding DNS server entries to the `/etc/resolv.conf` file.
- Editing the `/etc/mail/localhost.cf` file to enable the system to send messages using SMTP.
- Installing the bash package.

It should be noted that adding the bash package is not a task described in the OpenBSD afterboot manual page. However, it is convenient to install it at this point.

5.4.1 Modify the dot files in `/etc/skel`

The dot files are the default `.login`, `.profile`, `.cshrc`, `.mailrc` and `.rhosts` files that are copied to a user's home directory when a new user account is created. These files are located in the `/etc/skel` directory and are used to set up the user's shell environment.

One of the shell parameters is the PATH variable which is used to specify the directories that will be searched by the shell when the user attempts to execute a command. There is a problem with the standard `.profile` and `.cshrc` files because the PATH variable includes a '.' which means that the shell will search the current working directory for a command. This could make the user vulnerable to trojan horse programs if the system is compromised by an attacker. This could be really serious if the user is able to run commands as root using `sudo` because the user's PATH is searched and not the PATH in root's `.profile` or `.cshrc` file.

In the `.profile` file the dot at the end of the PATH variable declaration needs to be removed:

```
PATH=$HOME/bin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/
sbin:/usr/games:.
```

In the `.cshrc` file the dot at the end of the `set path` directive needs to be removed:

```
set path = (~/bin /bin /sbin /usr/{bin,sbin,local/bin,local/sbin,games}
.)
```

In addition, the .rhosts file can be removed because the Honeyd server will not be using the rsh, rcp or rlogin protocols.

Since this step is performed before any new user accounts are created it will not be necessary to modify `.profile` or `.cshrc` files elsewhere on the system. Note that the dot files in root's home directory (`/root`) are different to those in `/etc/skel`.

5.4.2 Install the bash Package

The bash package, which is installed purely for convenience, is available from CD #1. The following session transcript shows how to install it.

```
matrix# cd /
matrix# mkdir /cdrom
matrix# mount /dev/cd0a /cdrom
matrix# cd /cdrom/3.5/packages/i386
matrix# ls bash*
bash-2.05b-static.tgz
matrix# pkg_add bash*
Adding bash-2.05b-static.tgz

+---------------
| For proper use of bash-2.05b-static you should notify the system
| that /usr/local/bin/bash is a valid shell by adding it to the
| the file /etc/shells.  If you are unfamiliar with this file
| consult the shells(5) manual page
+---------------

matrix#
```

Despite the fact that the message directs us to manually add bash to the `/etc/shells` file, this can be left for the moment because it will be taken care of in the next task.

5.4.3 Create a User Account

A user account can be created using the `adduser` command. This is an interactive program that walks the administrator through the creation of a user account. The first time that `adduser` runs it prompts for default options to be configured.

OpenBSD 3.5 accepts passwords of up to 128 characters in length. Although somewhat extreme (unless you are the world memory champion and a very fast typist), this is a welcome change from some of the commercial versions of UNIX

that still limit the maximum number of characters to 8. In addition, OpenBSD offers the Blowfish algorithm which provides far stronger password encryption that DES, which is still used by most other UNIX systems. Although not listed as a risk mitigation step, the combination of Blowfish and longer passwords will be used in order to make it difficult for an individual to succeed with a password attack.

The following session transcript shows the use of `adduser`:

```
bash-2.05b# adduser
Couldn't find /etc/adduser.conf: creating a new adduser configuration
file
Reading /etc/shells
Found shell: /usr/local/bin/bash. Add to /etc/shells? (y/n) [y]: enter
Enter your default shell: bash csh ksh nologin sh [sh]: bash
Your default shell is: bash -> /usr/local/bin/bash
Reading /etc/login.conf
Default login class: auth-defaults auth-ftp-defaults daemon default
staff
[default]: enter
Enter your default HOME partition: [/home]: enter
Copy dot files from: /etc/skel no [/etc/skel]: enter
Send message from file: /etc/adduser.message no [no]: enter
Do not send message
Prompt for passwords by default (y/n) [y]: enter
Default encryption method for passwords: auto blowfish des md5 old
[auto]: blowfish
Use option ``-silent'' if you don't want to see all warnings and
questions.

Reading /etc/shells
Reading /etc/login.conf
Check /etc/master.passwd
Check /etc/group

Ok, let's go.
Don't worry about mistakes. I will give you the chance later to correct
any input.
Enter username []:smith
Enter full name []: Agent Smith
Enter shell bash csh ksh nologin sh [bash]: enter
Uid [1000]: enter
Login group smith [smith]: enter
Login group is ``smith''. Invite smith into other groups: guest no
[no]: wheel
Login class auth-defaults auth-ftp-defaults daemon default staff
[default]: enter
Enter password []:
Enter password again []:

Name:        smith
Password:    ****
Fullname:    Agent Smith
Uid:         1000
```

```
Gid:          1000 (smith)
Groups:       smith wheel
Login Class: default
HOME:         /home/smith
Shell:        /usr/local/bin/bash
OK? (y/n) [y]: enter
Added user ``smith''
Copy files from /etc/skel to /home/smith
Add another user? (y/n) [y]: n
Goodbye!
```

Note that my user ID was added to the wheel group in order to enable me to use `sudo` and `su` to root when necessary.

5.4.4 Edit the `/etc/sudoers` File

The final step in setting up the user environment involves editing the `/etc/sudoers` file to enable members of the wheel group to use sudo to run commands that require root privilege. sudo is configured to log all commands that it executes, thus providing an audit trail of actions taken on the system. Of course this relies on system administrators being disciplined enough to use `sudo` rather than `su`. The following line needs to be uncommented from the `sudoers` file:

```
%wheel   ALL=(ALL)        ALL
```

5.4.5 Edit the `/etc/resolv.conf` File

During the installation phase, a single DNS server was specified in the network configuration section. It is good practice to have more than one DNS server to call upon in case the primary server becomes unavailable. For this reason, the `/etc/resolv.conf` file needs to be edited. Once edited the file looks like this:

```
search fake.world.
nameserver 10.192.169.53
nameserver 10.160.169.53
nameserver 10.128.169.53
lookup file bind
```

5.4.6 Re-create the `/etc/mail/localhost.cf` File

Sendmail reads its configuration from the `/etc/mail/localhost.cf` file. Within this file the DS parameter is used to specify a smart host to which all messages are sent. It is possible to directly edit this file, but the problem with this approach is that if the localhost.cf file is recreated from the original `openbsd-`

`localhost.mc` macro file in `/usr/share/sendmail/cf`, this change will be lost. Therefore, the better approach is to add the following line to the bottom of the macro file:

```
define(`SMART_HOST',`mailhub.fake.world')
```

Once the macro file has been modified, the following steps need to be performed to create the new sendmail configuration file and move it to the appropriate directory:

```
cd /usr/share/sendmail/cf
sudo make openbsd-localhost.cf
sudo mv openbsd-localhost.cf /etc/mail/localhost.cf
```

The DS parameter in the resulting `localhost.cf` file will look like this:

```
# "Smart" relay host (may be null)
DSmailhub.fake.world
```

After the change has been made SIGHUP needs to be sent to sendmail in order to force it to re-read its configuration:

```
sudo kill -HUP `ps ax | grep sendmail | grep -v grep | \
awk '{ print $1 }'`
```

5.5 OS patches

Objective: This step is performed to satisfy RMS #4 and is intended to eliminate any known OS vulnerabilities that could potentially be exploited by an attacker.

Time required: 15 minutes (not including a kernel rebuild).

Since all of the base OS software has been installed, it is possible to proceed with patching of the system.

As mentioned in a previous section, OpenBSD patches are not distributed as binary files. Instead, they are distributed as diff files that can be applied to the source tree under `/usr/src`.

5.5.1 Patching Choices

There are three choices for patching OpenBSD:

1. Stick with the latest release and apply the patches by hand.

2. Use the *patch branch* which has the most important patches applied.

3. Use the *current* source for all the latest features.

Options 2 & 3 use CVS (Concurrent Versions System) in conjunction with RSH or SSH to obtain a source tree from one of the OpenBSD development team's CVS repositories. The source tree is then used to rebuild the system. Neither of these options are suitable for our environment because SSH access to the Internet is not currently available for systems connected to the organization's intranet or DMZ segments.

By default, our only option is to use the source tree supplied as a tarball on CD #3 and apply the patches by hand. The current list of patches can be found on the errata page[12] at the OpenBSD web site, along with a link to download a tar.gz file containing all the patches.

The steps to install the source tree are shown below:

```
sudo mount /dev/cd0a /cdrom
cd /usr/src
sudo tar xzf /cdrom/src.tar.gz
```

5.5.2 Apply Patches

Although an example of applying a patch[13] appears on the OpenBSD site, it is probably worth saying that the actual steps required will vary from patch to patch. In order to install a patch correctly, the associated instructions should be followed. The patches can be found at http://www.openbsd.org/errata.html. The following shows example instructions for applying a patch:

Apply by doing:
cd /usr/src
patch –p0 < 009_kerberos.patch

Rebuild and install the Kerberos 5 library:
cd lib/libkrb5
make obj
make depend
make
make install

And then rebuild and install the Kerberos 5 KDC:
cd ../../kerberosV/libexec/kdc

[12] http://www.openbsd.org/errata.html
[13] http://www.openbsd.org/faq/faq10.html#Patches

make obj
make depend
make
make install

In some instances a rebuild of the kernel and a system reboot are required after a patch has been applied to the source. The steps required to rebuild the default kernel and reboot the system with the new kernel are shown below.

```
cd /usr/src/sys/arch/i386/conf
sudo /usr/sbin/config GENERIC
cd /usr/src/sys/arch/i386/compile/GENERIC
sudo make clean
sudo make depend
sudo make
sudo cp /bsd /bsd.old
sudo cp bsd /bsd
sudo reboot
```

Note that all patching and rebuild tasks need to be executed with root privilege.

At the time of writing there are 12 patches for OpenBSD 3.5, however only 11 of these need to be applied to the Honeyd server because the X Windows software has not been installed. Even with this relatively small number, patching by hand could become tedious when all patches are applied together. Fortunately, several patches require a kernel rebuild so all the patches can be applied to the source tree before the rebuild takes place. In addition, there are 3 CVS related patches that can be applied to the source tree before performing the rebuild steps. Once the system is up-to-date, only incremental additions are required as new patches are released.

5.5.3 Patch Verification

Unlike other operating systems (e.g. Solaris with `showrev -p`), OpenBSD does not make it easy to determine which patches have been applied to the OS. For this reason, well defined maintenance procedures are needed to keep track of the patches that have been applied.

When faced with a system where no patching records have been kept, the source tree can be searched for the `*.orig` files that were created by the patch utility to backup the original source code files before the patch was applied. The following extract shows the result of a search on the Honeyd server immediately after all of the latest patches have been applied:

```
-bash-2.05b$ cd /usr/src
-bash-2.05b$ find . -name \*.orig -print
./gnu/usr.bin/cvs/lib/xsize.h.orig              ← 3 CVS patches
.
.
```

```
./kerberosV/src/kdc/config.c.orig                    ← 1 Kerberos patch
.
.
./sbin/isakmpd/ike_phase_1.c.orig                    ← 1 isakmpd patch
.
.
./sys/dev/ic/gdt_common.c.orig                       ← 1 gtd patch
.
.
./sys/miscfs/fifofs/fifo_vnops.c.orig                ← 1 FIFO patch
./sys/miscfs/procfs/procfs_cmdline.c.orig            ← 1 procfs patch
.
.
./sys/netinet/tcp_input.c.orig                       ← 1 TCP patch
./sys/scsi/scsi_base.c.orig                          ← 1 SCSI patch
.
.
./usr.sbin/httpd/src/include/http_core.h.orig        ← 1 httpd patch
.
.
                                          Total Patches: 11
```

Based on the above result it would appear that 11 patches have been applied to various parts of the source tree.

It should be noted that on some systems it may not be possible to tell if the source tree has been on the system since it was originally built or whether it has been recently created to enable the latest patches to be applied. Some site policies may not allow the source tree to be present on a production system, so it has to be temporarily added for each round of patch additions.

Patch verification is performed at this stage of the proceedings because the source tree will be removed from the system at a later stage as a best practice hardening step.

5.6 Services and Processes

Objective: Eliminating any unnecessary services is performed to satisfy RMS #8 and is intended to protect the system from vulnerabilities that have not yet been discovered.

Time required: 45 minutes.

5.6.1 Identify default services

The network services that are running by default can be determined by using the `netstat -an` command:

```
-bash-2.05b$ netstat -an
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
tcp        0     44  10.192.168.10.22       10.192.250.64.1852
ESTABLISHED
tcp        0      0  127.0.0.1.587          *.*
LISTEN
tcp        0      0  127.0.0.1.25           *.*
LISTEN
tcp        0      0  *.22                   *.*
LISTEN
tcp        0      0  *.37                   *.*
LISTEN
tcp        0      0  *.13                   *.*
LISTEN
tcp        0      0  *.113                  *.*
LISTEN
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
udp        0      0  127.0.0.1.512          *.*
udp        0      0  *.514                  *.*
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
tcp6       0      0  ::1.587                *.*
LISTEN
tcp6       0      0  ::1.25                 *.*
LISTEN
tcp6       0      0  *.22                   *.*
LISTEN
tcp6       0      0  *.37                   *.*
LISTEN
tcp6       0      0  *.13                   *.*
LISTEN
tcp6       0      0  *.113                  *.*
LISTEN
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
udp6       0      0  ::1.512                *.*
```


However, this is not that useful because it does not show any relationship between listening services and running processes. In order to get this information, the fstat command needs to be run as shown in the following session transcript:

```
matrix# fstat | grep internet
root     sendmail   30220    3* internet stream tcp 0xd0917b40
127.0.0.1:25
```

```
root     sendmail   30220    5* internet6 stream tcp 0xd0917ca8
[::1]:25
root     sendmail   30220    6* internet stream tcp 0xd0917e10
127.0.0.1:587
root     sendmail   30220    7* internet6 stream tcp 0xd0926004
[::1]:587
root     sshd       32513    4* internet6 stream tcp 0xd0917870 *:22
root     sshd       32513    5* internet stream tcp 0xd09179d8 *:22
root     inetd      20931    4* internet stream tcp 0xd0917000 *:113
root     inetd      20931    5* internet6 stream tcp 0xd0917168 *:113
root     inetd      20931    6* internet dgram udp 127.0.0.1:512
root     inetd      20931    7* internet6 dgram udp [::1]:512
root     inetd      20931    8* internet stream tcp 0xd09172d0 *:13
root     inetd      20931    9* internet6 stream tcp 0xd0917438 *:13
root     inetd      20931   10* internet stream tcp 0xd09175a0 *:37
root     inetd      20931   11* internet6 stream tcp 0xd0917708 *:37
_syslogd syslogd    29003    5* internet dgram udp *:514
matrix#
```

The output from netstat and fstat shows that the following services are listening on network sockets:

- **tcp 587**. OpenBSD 3.5 runs Sendmail 8.12.11 which implements a Message Submission agent as described in RFC 2476[14]. This service is bound to the loopback interface so it will not accept connections from other systems. This service is required by the Honeyd server so that messages and alerts can be sent to the appropriate parties.

- **tcp 25**. This is the SMTP service offered by Sendmail. This service is also bound to the loopback interface so it will not accept connections from other systems. This service is required by the Honeyd server so that messages and alerts can be sent to the appropriate parties.

- **tcp 22**. This is the SSH service offered by OpenSSH and is required for remote encrypted sessions (RMS #14).

- **tcp 37, tcp 13 & tcp 113**. These services were originally designed for network diagnostics and testing back in the days when the Internet was a friendlier place. Today, they are of very little use and could be used by an attacker to disrupt network services. Since these services are not required they will be eliminated in order to satisfy RMS #8.

- **udp 512**. This is the comsat service which will notify users that mail has arrived for them if they have biff enabled. Although the service is bound to the loopback interface, it is not required and will be disabled.

[14] http://www.faqs.org/rfcs/rfc2476.html

- **udp 514**. This port is used by syslogd to enable it to send log messages to another log server, which is the case because we need to satisfy RMS #20. Unless syslogd is started with the –u flag, it does not accept messages from udp port 514. Checking the `/etc/rc` and `/etc/rc.conf` files reveals that syslogd is not started with the –u flag. (As an aside, checking the process list with `ps ax` is not a reliable way to check for options being used because a process is able to change its description in the process list.)

To recap, only the SSH, SMTP, submission and syslog services should be running. All the others need to be removed. In addition, the system appears to be running IPv6 by default. Since this is not required it will also need to be removed.

5.6.2 Disable inetd

As can be seen from the following command output, all of the services that need to be eliminated are run in conjunction with the inetd server:

```
-bash-2.05b$ grep -v "^#" /etc/inetd.conf
ident           stream  tcp     nowait  _identd /usr/libexec/identd
identd -el
ident           stream  tcp6    nowait  _identd /usr/libexec/identd
identd -el
127.0.0.1:comsat dgram  udp     wait    root    /usr/libexec/comsat
comsat
[::1]:comsat    dgram   udp6    wait    root    /usr/libexec/comsat
comsat
daytime         stream  tcp     nowait  root    internal
daytime         stream  tcp6    nowait  root    internal
time            stream  tcp     nowait  root    internal
time            stream  tcp6    nowait  root    internal
```

Since none of the services that we wish to keep run out of inetd, the best approach to eliminating the unnecessary services is to delete the `/etc/inetd.conf` file and disable the inetd daemon. This can be achieved in four steps:

1. Kill the inetd daemon:

   ```
   sudo kill `ps ax | grep inetd | grep -v grep \
   | awk '{ print $1 }'`
   ```

2. Delete the `/etc/inetd.conf` file.

3. Edit the `/etc/rc.conf` file to prevent inetd starting at boot time by making the following change.

   ```
   #inetd=YES
   ```

```
inetd=NO.
```

4. Edit the `/etc/mtree/special` file and remove the following entry:

   ```
   inetd.conf      type=file mode=0644 uname=root gname=wheel
   ```

   This will prevent the `/etc/security` script from complaining about the file being absent.

Since the inetd daemon has been killed, the unwanted services will no longer be running.

5.6.3 Disable IPv6

IPv6 is not required by the Honeyd server so it will be disabled. In order to do this it is necessary to rebuild the kernel without the INET6 option. To disable IPv6 in the kernel, the following option needs to be commented out in the `/usr/src/sys/conf/GENERIC` file:

```
option          INET6           # Ipv6 (needs INET)
```

When the file has been modified, the kernel is rebuilt using the steps previously identified above in section 5.5.2, “Applying Patches”. However, before rebooting the server, some additional steps need to be performed.

A side effect of disabling IPv6 is that sendmail will not start because its configuration file contains directives to enable it to run with IPv6. There are two ways to correct this situation. Firstly, these directives can be commented out in the `/etc/mail/localhost.cf` file as shown below:

```
# SMTP daemon options

O DaemonPortOptions=Family=inet, address=127.0.0.1, Name=MTA
# O DaemonPortOptions=Family=inet6, address=::1, Name=MTA6, M=O
O DaemonPortOptions=Family=inet, address=127.0.0.1, Port=587, Name=MSA,
M=E
# O DaemonPortOptions=Family=inet6, address=::1, Port=587, Name=MSA6,
M=O, M=E
```

Again, the problem with this approach is that if the localhost.cf file is recreated from the original `openbsd-localhost.mc` macro file in `/usr/share/sendmail/cf`, these changes will be lost. Therefore, the better way to correct the situation is to comment out the relevant configuration parameters in the `openbsd-localhost.mc` macro file as follows:

```
DAEMON_OPTIONS(`Family=inet, address=127.0.0.1, Name=MTA')dnl
dnl DAEMON_OPTIONS(`Family=inet6, address=::1, Name=MTA6, M=O')dnl
```

```
DAEMON_OPTIONS(`Family=inet, address=127.0.0.1, Port=587, Name=MSA,
M=E')dnl
dnl DAEMON_OPTIONS(`Family=inet6, address=::1, Port=587, Name=MSA6,
M=O, M=E')dnl
```

Note that the “dnl” macro is required to make the m4 processor ignore a line (i.e. treat it as a comment). Once the macro file has been edited the following steps need to be performed to create the new sendmail configuration file and move it to the appropriate directory:

```
cd /usr/share/sendmail/cf
sudo make openbsd-localhost.cf
sudo mv openbsd-localhost.cf /etc/mail/localhost.cf
```

The /etc/hosts file needs to be edited to remove the following line, which relates to IPv6:

```
::1 localhost.aircanada.ca. localhost
```

The system can now be rebooted.

5.6.4 Identify running processes

The processes that are running can be identified using the `ps -ax` command:

```
  PID TT   STAT      TIME COMMAND
    1 ??  Is      0:00.02 /sbin/init
30062 ??  Is      0:00.01 syslogd: [priv] (syslogd)
 9449 ??  I       0:00.03 syslogd -a /var/empty/dev/log
18437 ??  Is      0:00.24 /usr/sbin/sshd
27335 ??  Is      0:00.01 cron
18853     Is      0:00.03 sendmail: accepting connections (sendmail)
21072 C0  Is+     0:00.01 /usr/libexec/getty Pc ttyC0
  981 C1  Is+     0:00.01 /usr/libexec/getty Pc ttyC1
18080 C2  Is+     0:00.01 /usr/libexec/getty Pc ttyC2
 3282 C3  Is+     0:00.04 /usr/libexec/getty Pc ttyC3
10789 C5  Is+     0:00.01 /usr/libexec/getty Pc ttyC5
```

This is not considered to be the list of default processes because `inetd` has already been disabled. However, since inetd is the only process that we have disabled, it can be seen that the default installation of OpenBSD 3.5 has very few processes running compared with Solaris or some Linux distributions. A brief description of each process is provided below:

- **`/sbin/init`**. init is the program that starts most of the daemons on the system.

- **`syslogd`**. This is the logging service and has already been mentioned. As of OpenBSD 3.4, syslogd was implemented as two processes. The child

process (PID 9449) runs with the privilege of a normal user and is chrooted in order to limit the effect of a successful exploit. The parent process (PID 30062) runs with root privilege and performs tasks on behalf of the child.

- **`/usr/sbin/sshd`**. sshd provides encrypted remote sessions and has already been discussed.

- **`cron`**. This daemon is used by the system to schedule tasks.

- **`sendmail`**. sendmail is the default SMTP messaging server. It may be somewhat disconcerting to see the sendmail entry in the process list with "accepting connections". However, as discussed already, sendmail only accepts connections from localhost.

- **`/usr/libexec/getty Pc ttyC[01235]`**. These five processes provide five virtual consoles on the machine and are controlled by the configuration in the `/etc/ttys` file. Note that there are no serial ports enabled by default (a process controlling a serial port looks like `/usr/libexec/getty std.9600 tty00`). In case you are wondering, ttyC4 is reversed for use with the X Window system.

5.6.5 Disable virtual consoles

The majority of the above processes are required. However, there is no need to have 5 virtual consoles on the system. The risk is that someone could forget to log out of one of the console sessions, thereby leaving the system wide open to unauthorized use. Although the Honeyd server will be located in a data centre, where physical access is strictly controlled, only one virtual console is needed so ttyC1, 2, 3 and 5 will be disabled. This is achieved by editing the /etc/ttys file as follows (changes are shown in bold):

```
#
#       $OpenBSD: ttys,v 1.17 2002/06/09 06:15:14 todd Exp $
#
# name  getty                           type    status
comments
#
console "/usr/libexec/getty Pc"         vt220   off secure
ttyC0   "/usr/libexec/getty Pc"         vt220   on  secure
ttyC1   "/usr/libexec/getty Pc"         vt220   off  secure
ttyC2   "/usr/libexec/getty Pc"         vt220   off  secure
ttyC3   "/usr/libexec/getty Pc"         vt220   off  secure
ttyC4   "/usr/libexec/getty Pc"         vt220   off secure
ttyC5   "/usr/libexec/getty Pc"         vt220   off  secure
```

In order to bring the change into effect, send a SIGHUP to init to force the `/etc/ttys` file to be re-read:

```
-bash-2.05b$ sudo kill -HUP 1
```

Note that the init process always has PID 1.

5.7 Logging

Objective: Configure the system to send log messages to a remote syslog server in order to satisfy RMS #20.

Time required: 5 minutes

The standard `/etc/syslog.conf` file is well set up for logging messages to local files on the system. However, when a system is compromised, one of the activities that an attacker will perform is to cover tracks by trying to clean the log files. In order to mitigate the risk of the losing valuable logging information, the syslogd configuration file needs to be modified so that all messages are also sent to one or more remote syslog severs.

5.7.1 Logging to a Remote Syslog Server

The `/etc/syslog.conf` file contains directives for sending messages to a loghost, but these are commented out by default:

```
# Uncomment to log to a central host named "loghost".   You need to run
# syslogd with the -u option on the remote host if you are using this.
# (This is also required to log info from things like routers and
# ISDN-equipment). If you run -u, you are vulnerable to syslog bombing,
# and should consider blocking external syslog packets
#*.notice;auth,authpriv,cron,ftp,kern,lpr,mail,user.none    @loghost
#kern.debug,user.info,syslog.info                           @loghost
#auth.info,authpriv.debug,daemon.info                       @loghost
```

Although the above filters that have been defined for remote logging are reasonably comprehensive, a better approach is to send all messages just in case.

The configuration steps that are needed to set up remote logging are as follows:

1. Add the following lines to `/etc/syslog.conf` configuration file to send all messages to three remote syslog servers:

```
*.*   @loghost1
*.*   @loghost2
```

```
*.*   @loghost3
```

2. Add the following entries to the /etc/hosts file:

```
10.192.170.51     loghost1
10.160.170.51     loghost2
10.128.170.51     loghost3
```

3. Send SIGHUP to the syslogd daemon to force it to re-read its configuration file:

```
sudo kill -HUP `cat /var/run/syslog.pid`
```

At this point it is prudent to check that log messages are being received at each syslog server.

5.8 Warning Banners

Objective: Follow best practice by implementing messages that provide warnings about the consequences of unauthorized use of the system.

Time required: 10 minutes

Access to the Honeyd server is achieved using either a console session at the machine or a remote session using SSH. In both cases, we need to be able to display a short warning message to potential users and abusers of the system.

Ideally, the warning banner should be displayed *before* an authorization attempt is made, but this is not always possible. In the case of SSH, this is relatively easy to do, but not all SSH clients support this capability. This is the reason for also displaying a banner *after* login.

5.8.1 Prior Warnings

In order to display a warning banner on the console screen, the following line needs to be modified in the `/etc/gettytab` file:

```
default:\
        :np:im=\r\n%s/%m (%h) (%t)\r\n\r\n:sp#1200:
```

should be changed to:

```
default:\
        :np:im=\r\n%s/%m (%h) (%t)\r\n\r\n\
For authorized uses only. The activities on this system are\r\n\
monitored. Evidence of unauthorized activities may be disclosed\r\n\
to the appropriate authorities.\r\n\r\n\
```

```
:sp#1200:
```

While the `/etc/gettytab` file is being edited, it is a good opportunity to make a modification that will cause the console screen to be cleared when a user logs out. In order to do this, modify the following line:

```
P|Pc|Pc console:\
        :np:sp#9600:
```

so that it becomes:

```
P|Pc|Pc console:\
        :np:sp#9600:\
        :cl=\E[H\E[2J:
```

These changes take effect immediately and will first be noticed the next time that a user logs out of the console.

The configuration changes required to display a warning banner before logging in with SSH is covered in the section entitled “OpenSSH”.

5.8.2 Subsequent Warnings

Displaying a warning message after a user has gained access to the system is easy. Firstly, the following code needs to be commented out of the `/etc/rc` file to prevent the system from displaying the OpenBSD version information that is part of the standard message of the day:

```
# patch /etc/motd
# if [ ! -f /etc/motd ]; then
#         install -c -o root -g wheel -m 664 /dev/null /etc/motd
# fi
# T=`mktemp /tmp/_motd.XXXXXXXXXX`
# if [ $? -eq 0 ]; then
#         sysctl -n kern.version | sed 1q > $T
#         echo "" >> $T
#         sed '1,/^$/d' < /etc/motd >> $T
#         cmp -s $T /etc/motd || cp $T /etc/motd
#         rm -f $T
# fi
```

Secondly, the contents of the `/etc/motd` file need to be replaced with an appropriate warning banner. In our case the following warning text is used:

```
For authorized uses only. The activities on this system are
monitored. Evidence of unauthorized activities may be disclosed
to the appropriate authorities.
```

Although not required as part of the risk mitigation steps for the Honeyd server, the above steps are considered to be best practice.

5.9 OpenSSH

Objective: Ensure that the requirements of RMS #14 are met and make changes necessary to improve the security of OpenSSH.

Time required: 5 minutes.

Since the use of the Compaq Proliant 5500 does not enable us to eliminate the need for a remote session service, RMS #14 requires encryption to be used for remote sessions. Although OpenBSD 3.5 ships with the latest version of OpenSSH (3.8.1 at the time of writing), a few configuration changes are necessary to improve security.

5.9.1 Configuration Changes

There are a number of best practices that can be applied to the default configuration of sshd, which is controlled by the `/etc/ssh/sshd_config` file.

The convention used in the default `sshd_config` file is to comment out the options that are using default values. The presence of all the configuration parameters makes it very easy to see how sshd is configured and options that have been changed can be easily identified because they are uncommented.

There are a number of changes that need to be made to the default `sshd_config` file that ships with OpenBSD 3.5

1. **Disable fallback to version** 1. Since there are some known security issues with SSH v1, it is best to prevent the server from falling back to version 1 of the protocol. In order to do this the following change needs to be made:

   ```
   #Protocol 2,1
   Protocol 2
   ```

2. **Prevent root login**. Allowing the superuser to log in directly using SSH is not a good idea for two reasons. From a system administration perspective we loose accountability since it may not be possible to tell who actually logged into the system. From a security perspective it means that if an unauthorized person obtained the root password they would be able to remotely access the Honeyd server with superuser privileges. By disabling this feature, an attacker in the possession of the root password would need access to the account of a normal user and that user account

would need to belong to the wheel group in order to `su` to root. To disable this feature the following change is made:

```
#PermitRootLogin yes
PermitRootLogin no
```

3. **Disable TCP port forwarding**. OpenSSH allows local and remote forwarding of TCP connections. This could be useful to an attacker who has compromised the system as it could enable them to circumvent firewall and other network access restrictions. To disable this feature the following change is made:

```
#AllowTcpForwarding yes
AllowTcpForwarding no
```

4. **Enable a banner**. As mentioned in the section on warning banners, we require a warning banner to be presented to a user before s/he logs into the server. It should be noted that not all SSH clients support this feature, which is why the warning banner is also displayed after a successful login. To enable this feature the following change is made:

```
#Banner /some/path
Banner /etc/motd
```

The ssh daemon should be sent a SIGHUP to force it to re-read the `/etc/sshd_conf` file in order to bring the above changes into effect.

5.10 NTP

Objective: Implement time synchronization using NTP as a best practice step in order to help support the remote logging requirement of RMS #20.

Time required: 30 minutes.

Although this task is not required to directly satisfy any of the risk mitigation steps, synchronizing the system clock with other systems on the network will help to make correlating log events that much easier. This is important because log messages will be sent to a remote log server in order to satisfy RMS #20. Implementing NTP is considered to be a best practice activity.

5.10.1 Installation

The NTP software can be built from the ports tree. The `make search` command can be used to determine which software is required:

```
-bash-2.05b$ cd /usr/ports/
-bash-2.05b$ make search key=ntp | more
Port:   ntp-4.1.1c
Path:   net/ntp/stable
Info:   network time protocol implementation
Maint:  Dan Harnett <danh@openbsd.org>
Index:  net
L-deps:
B-deps: :devel/autoconf/2.52 :devel/metaauto
R-deps:
Archs:  any
.
.
```

Before proceeding it is a good idea to check that there are no known vulnerabilities in the NTP software. Checking the vulnerabilities archive on the SecurityFocus web site does not reveal any known vulnerabilities in version 4.1.1, so we can proceed with obtaining and building the software.

Since the Honeyd server does not have direct Internet access, it is necessary to manually obtain the source code for the NTP software as well as any software that NTP is dependent upon. The location of the NTP source code can be found the `/usr/ports/net/ntp/Makefile.inc` file and the location of the autoconf 2.52 source code can be found in the `/usr/ports/devel/autoconf/2.52/Makfile` file. Once these files have been obtained, a directory called `distfiles` needs to be created under `/usr/ports` and the NTP and autoconf tarballs need to be copied to the `/usr/ports/distfiles` directory.

At this point the NTP software can be built and installed as follows:

```
cd /usr/ports/net/ntp/stable
sudo make install
```

5.10.2 Configuration

The next step is to create the `/etc/ntp.conf` file that will be read by the NTP daemon:

```
# synchronize from these NTP servers
server 10.192.171.123
server 10.160.171.123
server 10.128.171.123

# define access restrictions
restrict default ignore
restrict 10.192.171.123 nomodify noquery
restrict 10.160.171.123 nomodify noquery
```

```
restrict 10.128.171.123 nomodify noquery
restrict 127.0.0.1

driftfile /var/ntp/ntp.drift
```

By synchronizing from more than one source we can mitigate against the risk of an attacker trying to skew the system clock because the NTP daemon will throw out bogus time information. The access restrictions prevent an attacker from gleaning information (noquery) or modifying (nomodify) the running configuration of NTP.

Lastly, the /var/ntp directory needs to be created for the ntp.drift file:

```
sudo mkdir -p /var/ntp
```

Normally the `ntp.drift` file resides in the `/etc` directory. However, the intention is to set the system immutable flag on the `/etc` directory tree in order to prevent unauthorized changes to the system configuration. The `/var` directory tree will not have the immutable flag applied to it, so this is a logical place to locate the `ntp.drift` file.

In order to start NTP, it is necessary to reboot the system. The reason for this is that the tickadj program needs to be started before the kernel goes into secure mode otherwise it will not be able to adjust the system clock backwards.

The OpenBSD developers intended to have the `ntpd.pid` file created in the `/var/run` directory, but since there is a mistake in `rc.local` file, the `ntpd.pid` file does not get created at all. To correct this error the following change needs to be made:

```
if [ X"${ntpd}" == X"YES" -a -x /usr/local/sbin/ntpd \
    -a -e /etc/ntp.conf ]; then
        ntpd_flags="-p /var/run/ntpd.pid"
        if [ $securelevel -ge 1 ]; then
                #ntpd_flags="${ntpdflags} -x"
                ntpd_flags="${ntpd_flags} -x"
        fi
        echo -n ' ntpd';        /usr/local/sbin/ntpd ${ntpd_flags}
fi
```

5.11 pf

Objective: Implement IP-based access in order to satisfy RMS #9.

Time required: 10 minutes (assumes that the policy and filtering rules have already been created).

pf is the packet filter that has shipped with OpenBSD since version 3.0. If offers some very powerful and advanced features, but is relatively straightforward to configure.

The purpose of running pf is to provide IP-based access control to services running on the Honeyd server (RMS #9). On the surface it would appear that we only need to provide IP-based access control to SSH since it is the only service that is being offered by the Honeyd server. If this was the case, the TCP Wrappers program would suffice for this requirement. However, we would like to implement a more comprehensive access control policy in order to restrict packets to and from the Honeyd server.

5.11.1 IP-based Access Control Policy

The IP-based access control policy relates to both inbound and outbound packets.

For inbound packets to the server, the policy is:

- Allow connections from the management VLANs to the SSH service running on the Honeyd server.
- Block all other attempts to send packets directly to the Honeyd server.
- Allow all packets to pass through to the IP addresses of virtual hosts simulated by Honeyd.

For outbound packets from the server, the policy is:

- Allow outbound packets from the Honeyd server for ICMP echo-request, SSH, SMTP, DNS, NTP and syslog.
- Block all other outbound packets from the Honeyd server.
- Allow all packets to pass through from the IP addresses of virtual hosts simulated by Honeyd.

The above policy is very general with respect to the connectivity requirements of the virtual hosts that will be simulated by Honeyd. It may need to be modified for some Honeyd configurations.

In addition, a pf feature called *state modulation* will be used to prevent passive fingerprinting of the Honeyd server. This can only be applied to TCP packets because the technique is based on introducing randomness into TCP sequence numbers in order to mask the normal TCP sequence number characteristics of the operating system.

The following rules in the `/etc/pf.conf` file implement the policy described above:

```
#       $OpenBSD: pf.conf,v 1.27 2004/03/02 20:13:55 cedric Exp $
#
# See pf.conf(5) and /usr/share/pf for syntax and examples.

# create a macro for the interface so that the rules will still
# apply even if the ip address of the host changes.
IF="fxp0"

# create macros for management VLANs and specific hosts
MAN_VLAN="{ 10.192.250.0/24, 10.160.250.0/24, 10.128.250.0/24 }"
SMTP="{ 10.192.172.25 }"
DNS="{ 10.192.169.53, 10.160.169.53, 10.128.169.53 }"
NTP="{ 10.192.171.123, 10.160.171.123, 10.128.171.123 }"
SYSLOG="{ 10.192.170.51, 10.160.170.51, 10.128.170.51 }"

# allow all traffic on the loopback interface
pass quick on lo0 all

########################
# Inbound connections #
########################

# only allow ssh connections to honeyd server from management
# VLANs
pass in log quick proto tcp from $MAN_VLAN to $IF port = 22 flags S/SA
keep state

# allow traffic from any address to any address except the
# honeyd server's address to enable Honeyd to work.
pass in log quick from any to ! $IF keep state

# default deny for inbound connections
block drop in log all

########################
# Outbound connections #
########################

# allow icmp echo requests to hosts on the same LAN
pass out log quick proto icmp from $IF to $IF:network icmp-type 8 code
0 keep state

# Use the modulate state feature of pf to make it harder to use
# passive fingerprinting against the honeyd host. However, this
# should only be done for connections that originate from the
# honeyd host and not for connections that originate from the
# virtual hosts within honeyd.

# allow all ssh connections from the honeyd host
pass out log quick proto tcp from $IF to any port = 22 flags S/SA
modulate state

# allow all smtp connections from the honeyd host
pass out log quick proto tcp from $IF to $SMTP port = 25 flags S/SA
modulate state
```

```
# allow dns packets from the honeyd host to the specified dns servers
pass out quick proto udp from $IF to $DNS port = 53 keep state
pass out quick proto tcp from $IF to $DNS port = 53 modulate state

# allow ntp packets from the honeyd host to the specified ntp servers
pass out quick proto udp from $IF to $NTP port = 123 keep state

# allow syslog packets from the honeyd host to the specified
# syslog server
pass out quick proto udp from $IF to $SYSLOG port = 514 keep state

# deny all other connections that originate from the honeyd host
# itself
block drop out log quick from $IF to any

# explicitly allow all connections from the virtual hosts within
# Honeyd
pass out log all keep state

###################
# End of ruleset #
###################
```

5.11.2 Enable pf and pflogd

Once the `/etc/pf.conf` file has been created the following steps need to be performed in order to enable pf and the logging daemon, *pflogd*.

1. The following command should be used to verify the syntax of the rules in the `/etc/pf.conf` file whenever changes are made to it (this includes when the file is initially created):

   ```
   sudo pfctl -n -f /etc/pf.conf
   ```

2. Enable pf with the following command:

   ```
   sudo pfctl -e -f /etc/pf.conf
   ```

   If this task is performed remotely it will result in the SSH session to the Honeyd server being dropped because pf did not see the initial packet containing the SYN which would cause a match of the inbound SSH connection rule. Since there is no entry in the state table for the SSH session, all SSH packets are dropped.

3. If these steps are not being performed at the system console, re-establish the SSH session and run the following command to bring up the pflog0 interface:

```
sudo ifconfig pflog0 up
```

The reason for doing this is because pflogd makes the logging data available on the pflog0 interface so that it can be examined in real time with tcpdump.

4. Now start the pflogd daemon with the following command:

```
sudo pflogd
```

5. In order to make pf and pflogd start when the system boots, the pf parameter needs to be set to “YES” in the /etc/rc.conf file as follows:

```
pf=YES                  # Packet filter / NAT
```

6. Reboot the server and check that pf is operating after the reboot.

5.12 Honeyd

Objective: Install and configure Honeyd and sandbox it with systrace in order to satisfy the requirements of RMS #6.

Time required: 90 minutes.

As noted in the server specification section of this document, Honeyd depends upon and works in conjunction with several other software components. In the following description of the software installation process it is assumed that the source code for all of these programs has been obtained from their respective sources and has been transferred to the Honeyd server. This includes moving the appropriate files to the `distfiles` directory in the ports tree:

```
cd /home/smith
sudo cp libtool-1.3.5.tar.gz libdnet-1.7.tar.gz \
autoconf-2.13.tar.gz /usr/ports/distfiles/
```

5.12.1 Installation

The following steps need to be performed in order to install the software components.

1. Build and install libtool:

```
cd /usr/ports/devel/libtool
```

```
sudo make install
```

2. Build and install libdnet:

```
cd /usr/ports/net/libdnet
sudo make install
```

3. Build and install autoconf 2.13:

```
cd /usr/ports/devel/autoconf/2.13
sudo make install
```

4. Build and install Honeyd:

```
cd /tmp
cp /home/smith/honeyd-0.8b.tar.gz .
tar xzf honeyd-0.8b.tar.gz
cd honeyd-0.8b
./configure
make
sudo make install
cp -R scripts/ /usr/local/share/honeyd/
```

5. Build and install libevent:

```
cd /tmp
cp /home/smith/libevent-0.8a.tar.gz .
tar xzf libevent-0.8a.tar.gz
cd /libevent-0.8a
./configure
make
sudo make install
```

6. Build and install arpd (note that it is necessary to specify the location of the version of libevent that was installed in the previous step in order to avoid running into problems with using the version of libevent that is shipped with OpenBSD):

```
cd /tmp
cp /home/smith/arpd-0.2.tar.gz .
tar xzf arpd-0.8.tar.gz
cd arpd
./configure --with-libevent=/usr/local --with-libdnet=/usr/local
make
sudo make install
```

It should be noted that the above steps will install the basic software that will get Honeyd up and running. When the details of a Honeyd deployment are finalized it may be necessary to add custom scripts to Honeyd in order to meet the requirements of a particular worm defense strategy. Although the details relating

to a particular Honey deployment are beyond the scope of this document, the process required to make changes will be covered in section 6, “Ongoing Maintenance Procedures”.

5.12.2 Start-up Scripts

When third party software is added to an OpenBSD system it is normal to modify the `/etc/rc.local` and `/etc/rc.conf.local` files so that the appropriate processes are started when the system boots up. For the Honeyd server, arpd and honeyd processes need to be started at boot time.

The `/etc/rc.conf.local` file needs to be created and the following shell script added to it:

```
#!/bin/sh -
#
# rc.local.conf for matrix.fake.world

# This is the safest way to do it because if the rc.local file does not
# see the arpd variable it will not start arpd. Starting arpd without
# specifying an IP address means that it will answer for all
# unallocated IP addresses on the LAN. This is not desirable.
arpd=YES
honeyd=YES

# specify the IP addresses here
arpd_flags="10.192.168.11"     # change to correspond with virtual IPs
being simulated
honeyd_flags="10.192.168.11"    # change to correspond with virtual IPs
being simulated
```

The following needs to be added to the `/etc/rc.local` file:

```
if [ X"${arpd}" == X"YES" -a -x /usr/local/sbin/arpd ]; then
        echo -n ' arpd';       /usr/local/sbin/arpd ${arpd_flags} \
                               > /dev/null 2>&1
fi

if [ X"${honeyd}" == X"YES" -a -x /usr/local/bin/honeyd \
    -a -e /etc/honeyd.conf ]; then
        saved_flag="${honeyd_flags}"
        honeyd_flags="-f /etc/honeyd.conf"
        if [ -e /usr/local/share/honeyd/nmap.prints ]; then
                honeyd_flags="${honeyd_flags} \
             -p /usr/local/share/honeyd/nmap.prints"
        fi
        if [ -e /usr/local/share/honeyd/xprobe2.conf ]; then
                honeyd_flags="${honeyd_flags} \
             -x /usr/local/share/honeyd/xprobe2.conf"
        fi
        if [ -e /usr/local/share/honeyd/nmap.assoc ]; then
                honeyd_flags="${honeyd_flags} \
```

```
            -a /usr/local/share/honeyd/nmap.assoc"
        fi
        if [ -e /usr/local/share/honeyd/pf.os ]; then
            honeyd_flags="${honeyd_flags} \
            -0 /usr/local/share/honeyd/pf.os"
        fi
        honeyd_flags="${honeyd_flags} ${saved_flag}"
        echo -n ' honeyd';       /usr/local/bin/honeyd ${honeyd_flags}\
                                 > /dev/null 2>&1
fi
```

5.12.3 Verification of Basic Honeyd Operation

In order to verify that Honeyd works correctly, it is necessary to create a very basic configuration. The following configuration creates a single virtual host that will be simulated by Honeyd:

```
# Test configuration for Honeyd server
create router
set router personality "Cisco 3600 router running IOS 12.2(6c)"
set router default tcp action reset
set router default udp action reset
set router default icmp action open
add router tcp port 23 "/usr/local/share/honeyd/scripts/router-
telnet.pl"
bind 10.192.168.11 router
```

The following session transcript shows the commands needed to run arpd and honeyd along with the resulting output.

```
-bash-2.05b$ sudo arpd 10.192.168.11
arpd[12395]: listening on fxp0: arp and (dst 10.192.168.11) and not
ether src 00:50:8b:0e:a1:2f
-bash-2.05b$ sudo honeyd -p /usr/local/share/honeyd/nmap.prints \
-f /etc/honeyd.conf -x / usr/local/share/honeyd/xprobe2.conf \
-a / usr/local/share/honeyd/nmap.assoc \
-0 / usr/local/share/honeyd/pf.os 10.192.168.11
Honeyd V0.8b Copyright (c) 2002-2004 Niels Provos
honeyd[24174]: started with -p /etc/nmap.prints -f /etc/honeyd.conf -x
/etc/xprobe2.prints -a /etc/nmap.assoc -0 / usr/local/share/honeyd/pf.os
10.192.168.11
Warning: Impossible SI range in Class fingerprint "IBM OS/400 V4R2M0"
Warning: Impossible SI range in Class fingerprint "Windows NT 4 SP3"
honeyd[24174]: listening promiscuously on fxp0: (arp or ip proto 47 or
(ip and (host 10.192.168.11))) and not ether src 00:50:8b:0e:a1:2f
Honeyd starting as background process
```

In order to verify its operation it is necessary to interact with the virtual host. The following session transcript shows ping and TCP tests being performed from another system.

```
-bash-2.05b$ ping -c 5 10.192.168.11
PING 10.192.168.11 (10.192.168.11): 56 data bytes
64 bytes from 10.192.168.11: icmp_seq=1 ttl=64 time=1.224 ms
64 bytes from 10.192.168.11: icmp_seq=2 ttl=64 time=0.380 ms
64 bytes from 10.192.168.11: icmp_seq=3 ttl=64 time=0.386 ms
64 bytes from 10.192.168.11: icmp_seq=4 ttl=64 time=0.380 ms
--- 10.192.168.11 ping statistics ---
5 packets transmitted, 4 packets received, 20.0% packet loss
round-trip min/avg/max/std-dev = 0.380/0.592/1.224/0.365 ms
-bash-2.05b$ telnet 10.192.168.11
Trying 10.192.168.11...
Connected to 10.192.168.11.
Escape character is '^]'.


For authorized uses only. The activities on this system are
monitored. Evidence of unauthorized activities may be disclosed
to the appropriate authorities.


 User Access Verification

Username: admin
Password:
% Access denied

Username: admin
Password:
% Access denied

Username: cisco
Password:
% Access denied
Connection closed by foreign host.
-bash-2.05b$
```

Based on these tests, it appears that our simple Honeyd configuration is working as expected.

5.12.4 Sandboxing Honeyd

In order to satisfy RMS #6, Honeyd needs to be run in a sandbox in order to provide protection in the event that an attacker finds vulnerabilities in Honeyd that could lead to the compromise of the Honeyd server.

The recommended options for sandboxing Honeyd are either to set up a chroot jail or to use a tool called *systrace*[15]. systrace does not run on all operating systems, but as the Honeyd server is running OpenBSD, the approach taken here is to use systrace because it offers more control of Honeyd than a traditional chroot jail would.

[15] http://www.citi.umich.edu/u/provos/systrace/

systrace is another creation of Niels Provos and was written as a policy enforcement tool capable of monitoring, intercepting and restricting system calls. systrace acts as a wrapper for a program and controls the environment in which the program operates. Control is exercised in accord with the policy for the program, so if the program tries to deviate from its normal behavior – for example, if an attacker tries to exploit vulnerabilities – systrace should prevent it from doing so.

In order for systrace to be able to enforce a policy for Honeyd, a policy needs to be created. The easiest way to do this is to let systrace automatically generate it by starting systrace with the –A flag set:

```
-bash-2.05b# systrace -A /usr/local/bin/honeyd \
             -f /etc/honeyd.conf \
             -p /usr/local/share/honeyd/nmap.prints \
             -a /usr/local/share/honeyd/nmap.assoc \
             -x /usr/local/share/honeyd/xprobe2.conf \
             -0 / usr/local/share/honeyd/pf.os \
             10.192.168.11
Honeyd V0.8b Copyright (c) 2002-2004 Niels Provos
honeyd[2179]: started with -f /etc/honeyd.conf -p
/usr/local/share/honeyd/nmap.prints -a
/usr/local/share/honeyd/nmap.assoc -x
/usr/local/share/honeyd/xprobe2.conf -0 / usr/local/share/honeyd/pf.os
10.192.168.11
Warning: Impossible SI range in Class fingerprint "IBM OS/400 V4R2M0"
Warning: Impossible SI range in Class fingerprint "Microsoft Windows NT
4.0 SP3"
honeyd[2179]: listening promiscuously on fxp0: (arp or ip proto 47 or
(ip and (host 10.192.168.11))) and not ether src 00:50:8b:0e:a1:2f
Honeyd starting as background process
```

Note that because systrace was started using the root account the resulting policy file(s) will be located in the `/root/.systrace` directory.

From a security perspective it is very important that the Honeyd executable is trusted because the resulting policy will be used to protect the system when systrace is run in policy enforcement mode. That is, if we generate policy for a trojaned executable, systrace may not be able to prevent the trojan code from performing its intended purpose. Therefore, it is essential to review the policy before putting it into production to ensure that there is no unexpected behavior. Of course, this requires in-depth knowledge of the application and of system calls.

From a functional perspective it is important to fully exercise the application (Honeyd in this case) so that its full range of behavior is observed by systrace and the corresponding policy can be generated. Again, it is important to review the policy before putting into production so that any fine tuning can be performed.

For the purpose of this exercise, we'll move forward with the simple Honeyd configuration that was used for verification purposes. Since Honeyd is only simulating one virtual system, it is fairly easy to exercise its full range of behavior:

1. Perform a ping test.
2. Connect and interact with the simulated telnet service.
3. Attempt a TCP connection to an unused port.
4. Send a UDP packet to an unused port.

Once these tests have been performed, the Honeyd process should be killed so that it exits normally. Since systrace was run from the root account, the resulting policy will be located in a file called `usr_local_bin_honeyd` the `/root/.systrace` directory. In addition to the policy for Honeyd, there is a separate policy for the router-telnet.pl script in a file called `usr_local_share_honeyd_scripts_router_telnet_pl`. These policies are included in Appendix A.

Review of these policies shows that Honeyd and the accompanying Perl script use system calls in the expected manner. Of course the policy for Honeyd only applies to the simple configuration that is being considered here. The full production scenario is likely to be much more complicated and the appropriate policy could take considerable time to refine. There are also likely to be several other policy files corresponding to the service scripts needed by the various virtual systems.

Now that the required systrace policy has been generated, the following steps need to be performed (as root) to enforce the policy.

1. Move the policy files to the `/etc/systrace` directory:

```
cd /root/.systrace
mv usr_local_bin_honeyd /etc/systrace
mv usr_local_share_honeyd_scripts_router_telnet_pl /etc/systrace
```

2. Modify the start-up script in the `/etc/rc.local` file so that Honeyd is run under systrace:

```
#echo -n ' honeyd'; /usr/local/bin/honeyd ${honeyd_flags} \
#                   > /dev/null 2>&1
echo -n ' honeyd';  /bin/systrace -a \
                    /usr/local/bin/honeyd ${honeyd_flags} \
                    > /dev/null 2>&1
```

Note that the `router-telnet.pl` script will automatically be run under systrace because it is a child of the honeyd process.

3. Although it is possible to start Honeyd under systrace by launching it from the command line, it is prudent to reboot the system in order to check that the rc script works correctly. When the system has completed the boot up sequence, the `ps ax` command can be used to verify that the systrace and honeyd processes are running:

```
28701 ??  Is      0:00.04 systrace -a honeyd -f /etc/honeyd.conf
-p /usr/local/share/honeyd/nmap.prints -a /usr/local/share
27963 ??  Ixs     0:00.01 honeyd -f /etc/honeyd.conf -p
/usr/local/share/honeyd/nmap.prints -a
/usr/local/share/honeyd/nmap
```

This concludes the installation and configuration of the Honeyd software.

5.13 System Heartbeat

Objective: Implement a simple heartbeat mechanism to satisfy the requirement of RMS #25.

Time required: 10 minutes.

The purpose of the system heartbeat is to verify the operation of the mechanisms that are used to send reports and alerts. The mechanisms that need to be tested are:

1. SMTP mail.

2. Logging to a remote syslog server and the subsequent generation of an alert from that server.

In both cases, the heartbeat will be sent every five minutes to the operations group who provide 24x7x365 coverage.

5.13.1 Piggybacking System Checks

In order to check for the appearance of new services and processes on the Honeyd server, we will piggyback output from the netstat and ps commands onto the SMTP Heartbeat message.

The following command will be used to check for new UDP and TCP services:

```
netstat -an | grep LISTEN
```

The following command will be used to check for the addition of new processes:

```
ps ax
```

5.13.2 SMTP Heartbeat

The SMTP heartbeat is implemented by adding the following line to root's crontab file:

```
*/5  *  *  *  *  (netstat -an | perl -ple 'last if /Active UNIX domain
sockets/'; echo; ps ax) | mail -s "SMTP Heartbeat from `/bin/hostname`"
secops@fake.world
```

This cron job will run every five minutes.

On the receiving system at the operations console an alert will be generated if:

1. The resulting message has not arrived within one minute of the time that it should have been sent.

2. There are any abnormalities in the output from netstat or ps.

It should be noted that the list of processes will vary because of scheduled jobs running from cron and programs being spawned by Honeyd during the course of its operation. The monitoring scripts on the operations console are designed to take these variations into account. A typical heartbeat message would look like this:

```
Subject: matrix.fake.world. SMTP Heartbeat
   Date: Wed, 21 Jul 2004 16:25:02 +0000 (GMT)
   From: Charlie Root <root@fake.world>
     To: secops@fake.world



Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
ip         0      0  *.*                    *.*                    255
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
tcp        0      0  127.0.0.1.587          *.*
LISTEN
tcp        0      0  127.0.0.1.25           *.*
LISTEN
tcp        0      0  *.22                   *.*
LISTEN
```

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
udp        0      0  *.*                    *.*
udp        0      0  *.*                    *.*
udp        0      0  10.192.168.10.123      *.*
udp        0      0  127.0.0.1.123          *.*
udp        0      0  *.123                  *.*
udp        0      0  *.514                  *.*

  PID TT   STAT      TIME COMMAND
    1 ??  Is      0:00.02 /sbin/init
26739 ??  Is      0:00.03 syslogd: [priv] (syslogd)
20742 ??  I       0:00.92 syslogd -a /var/empty/dev/log
 3045 ??  Is      0:00.01 pflogd: [priv] (pflogd)
32751 ??  I       0:11.08 pflogd: [running] -s 116 -f /var/log/pflog
(pflogd)
 8201 ??  Is      0:00.01 /usr/sbin/sshd
 7974 ??  Is      0:10.41 sendmail: accepting connections (sendmail)
30424 ??  Is      0:11.95 /usr/local/sbin/ntpd -p /var/run/ntpd.pid -x
15627 ??  Is      0:01.29 /usr/local/sbin/arpd 10.192.168.11
21488 ??  Is      0:00.02 /bin/systrace -a /usr/local/bin/honeyd -f
/etc/honeyd
28443 ??  Ixs     0:05.12 /usr/local/bin/honeyd -f /etc/honeyd.conf -p
/usr/loc
29269 ??  Is      0:01.50 cron
8802 ??  I       0:00.00 cron: running job (cron)
18516 ??  Is      0:00.01 /bin/sh -c /bin/sh
14991 ??  R       0:00.00 (hostname)
13978 ??  I       0:00.00 mail -s matrix.fake.world. SMTP Heartbeat
secops
18025 ??  I       0:00.00 /bin/sh -c /bin/sh
31978 ??  R       0:00.00 ps -ax
 1401 p0  Is      0:00.02 -bash (bash)
22093 p0  I       0:00.02 -csh (csh)
10407 p0  I+      0:00.05 bash
10265 C0  Is+     0:00.01 /usr/libexec/getty Pc ttyC0
```

5.13.3 Syslog Heartbeat

The syslog heartbeat is implemented by adding the following line to root's crontab file:

```
*/5  *  *  *  *  /usr/bin/logger -p user.notice \
-t root syslog heartbeat
```

This cron job will run every five minutes and will result in the following log message being sent to the remote syslog server:

```
Jul  5 17:10:01 matrix root: syslog heartbeat
```

On the remote syslog server a program called swatch is running, which is configured to send an alert to the operations group when it sees the "syslog heartbeat" entry in the `/var/log/messages` file. If the resulting email alert message does not arrive at the operations console within two minutes of the time at which it should have been sent, an alert will be generated.

5.14 File Integrity Checker

Objective: Implement a file integrity checker to detect changes to essential system files and satisfy RMS #18.

Time required: 45 minutes.

OpenBSD ships with a directory hierarchy mapping tool called mtree[16]. In mapping the structure of a directory tree, mtree produces a specification that records key characteristics of files and directories that are found. By supplying the appropriate command line switches, mtree can generate a message digest for each file, which means that it has the basic functionality required for performing file integrity checks.

In order to implement a file integrity checker with mtree the following issues need to be addressed:

- Determining the files and directories that need to be checked.
- Generating the specification files.
- Storing the specification files.
- Checking for changes.

5.14.1 Mapping Files and Directories

The layout of the file system works to our advantage because mtree can be prevented from descending beyond mount points using the '-x' switch. Careful selection of the root of the file hierarchy to be mapped will cause mtree to group some directories together within a single mapping activity.

Before determining which directories need to be mapped, it is a good idea to identify which directories will not be subject to file integrity checks:

- **`/var`**. This directory hierarchy contains *variable* data and would be difficult to monitor in a meaningful fashion. The exception is the `/var/cron/tabs`

[16] http://www.openbsd.org/cgi-bin/man.cgi?query=mtree

directory which contains the crontab files, including the one belonging to root.

- **/home**. Although there will not be many normal user accounts on the Honeyd server, directories below /home will be used by system administrators to transfer files to and from the system.

- **/usr/src**. The mapping of the source tree would be time consuming and would generate a large specification file. Since the source tree is only required for applying patches it would be best to store it as a tarball on another system and remove it from the Honeyd server.

- **/usr/ports**. Although not as large as the source tree, the ports tree would also generate a large specification file. Since the ports trees is only required when adding new software packages it would be best to store it as a tarball on another system and remove it from the Honeyd server.

The specification files needed for checking the remaining files and directories of interest can be generated with the following invocations of mtree:

1. **Rooted at** /. This will include files in /etc, /boot, /bin and /sbin as well as the kernel itself (/bsd), but will not go beyond the /usr, /home, /var or /tmp mount points. Mapping time is 5 seconds and the specification file size is approximately 125 Kbytes.

2. **Rooted at /dev**. Although mtree will map the /dev tree when rooted at /, problems will be experienced because the modification time, uid and gid values will change as part of normal system operation. Therefore, a separate mapping activity is necessary to collect only values of attributes that remain static. Mapping time is 1 second and the specification file size is approximately 70 Kbytes.

3. **Rooted at /usr**. This will include important binary files such as those in /usr/bin, /usr/sbin and /usr/libexec, but will not go beyond the /usr/src and /usr/ports or /usr/local mount points. Mapping time is 47 seconds and the specification file size is approximately 2.8 Mbytes.

4. **Rooted at /usr/local**. This will include binary files such as those in /usr/local/bin, /usr/local/sbin and /usr/local/libexec. Mapping time is 2 seconds and the specification file size is approximately 44 Kbytes.

5. **Rooted at /var/cron/tabs**. This directory contains the crontab files, includes the one belonging to root. Mapping time is less than 1 second and the specification file size is approximately 480 bytes.

By default, mtree records the following attributes of files and directories, where applicable:

- **gid**. Group ownership of the file.
- **mode**. The file permissions.
- **nlink**. The number of hard links to the file.
- **size**. The size of the file.
- **link**. The file referenced by the symbolic link, if applicable.
- **time**. The modification time of the file.
- **uid**. The file owner.

When combined with message digests, it is felt that this information is sufficient to be able to determine the type of change that could be made to a file or a directory.

When mapping `/dev`, the gid, uid, mode and time attributes will not be collected because they are subject to change as part of normal system operation.

In order to mitigate the risk of an attacker being able to modify a system file in a way that would make the resulting message digest identical to that of the original file, two message digests will be created for each file using the MD5 and SHA-1 algorithms. Although hash collision is possible with MD5, it will be used because of the relatively low computational overhead. The SHA-1 algorithm produces a longer message digest than MD5 and is generally thought to be less prone to hash collision problems. In combination, the two digests strike a good balance between security and computational burden.

5.14.2 Generation of Specification Files

Before any specification files are generated, the source and ports trees need to be saved and removed from the system. This is a best practice step and aims to protect the integrity of the source files that we have invested time in modifying.

1. Create tarball of `/usr/src` and transfer it to another system.

```
cd /usr/src/
tar czf /home/smith/matrix_src.tar.gz .
cd /home/smith
```

```
scp matrix_src.tar.gz backup@vault.fake.world:.
rm matrix_src.tar.gz
```

2. Delete the source tree.

```
cd /
umount /usr/src
newfs /dev/rsd0f
mount /dev/sd0f /usr/src
```

3. Create tarball of `/usr/ports` and transfer it to another system.

```
cd /usr
tar czf /home/smith/matrix_ports.tar.gz ports
cd /home/smith
scp matrix_ports.tar.gz smith@vault.fake.world:.
rm matrix_ports.tar.gz
```

4. Delete the ports tree.

```
cd /
umount /usr/ports
newfs /dev/rsd0e
mount /dev/sd0e /usr/ports
```

The commands used to generate the specification files are as follows:

```
mtree -cx -K md5digest,sha1digest -p / > /tmp/root.spec
mtree -cx -k nlink,size,link -K md5digest,sha1digest -p /dev >
/tmp/dev.spec
mtree -cx -K md5digest,sha1digest -p /usr > /tmp/usr.spec
mtree -cx -K md5digest,sha1digest -p /usr/local > /tmp/usr_local.spec
mtree -cx -K md5digest,sha1digest -p /var/cron/tabs > /tmp/crontab.spec
```

It should be noted that the specification file for the hierarchy rooted at `/` (`root.spec`) will need to be modified so that certain attribute/value pairs are excluded in order to prevent unnecessary alerts being generated:

1. The `/dev` directory and its entire contents can be ignored because it will be mapped separately.

2. Setting a root of `/` will cause mtree to map `/var`, `/tmp`, `/floppy`, `/mnt` and `/cdrom` mount points. Since the value of the modification time attribute will change for these directories in the course of normal system operation, this attribute/value pair needs to be removed from the specification of each of these directories.

3. The `/root/.bash_history` file's modification time, size and MD5 and SHA-1 hash values will change from time to time, so all of these attribute/value pairs need to be removed for the `root.spec` file.

In the `crontab.spec` file, the modification time may change for the `/var/cron/tabs` directory and `/var/cron/tabs/.sock` file. Therefore, the attribute/value pair will need to be removed for these items.

The following Perl script (`/usr/local/sbin/mod-mtree-spec`) automates the process of modifying the `root.spec` and `crontab.spec` files:

```
#!/usr/bin/perl -w

use strict;

my @dirs = ("/cdrom", "/dev", "/floppy", "/mnt", "/root", "/tmp",
"/var");
foreach (@dirs) {
    $_ = "^# .$_\$";
}
my $pattern = join "|", @dirs;
my @found = ();
my $bash_lines = 0;
my $crontab = 0;

while (<>) {
    chomp;
    $crontab++ if m%^#\s+tree:\s+/var/cron/tabs%;
    if ($crontab) {
        s/(^\.\s+type=\w+\s+mode=\d+\s+nlink=\d+)\s+time=\d+\.\d+/$1/;
        s/(^\s+\.sock\s+type=\w+)\s+time=\d+\.\d+/$1/;
    }
    unshift @found, $_ if m%$pattern%;
    if (@found % 2 == 1) {
        s/time=\d+\.\d+//;
        if ($found[0] =~ m%/dev%) {
            s/(^dev\s+type=dir.+)/$1 ignore/;
            unless ( m%\\% || m%^#% || m%\.\.% || m%/set%) {
                s/(.+)/$1 optional/;
            }
        }
        if ($found[0] =~ m%/root%) {
            if (/\.bash_history/) {
                $bash_lines++;
            }
            elsif ($bash_lines && $bash_lines <= 3 &&
/mode=\d+\s+size=\d+/){
                s/(mode=\d+)\s+size=\d+.+$/$1/;
                $bash_lines++;
            }
            elsif ($bash_lines && $bash_lines <= 3) {
                $bash_lines++;
                next;
```

```
            }
        }
    }
    print "$_\n";
}

exit 0
```

The script is intended to be used in the following manner:

```
cd /tmp
/usr/local/sbin/mod-mtree-spec root.spec > root.spec.new
mv root.spec.new root.spec
/usr/local/sbin/mod-mtree-spec crontab.spec > crontab.spec.new
mv crontab.spec.new crontab.spec
```

After running the `mod-mtree-spec` script on the `root.spec` file, the effect on the `/tmp` entry, for example, would be to remove the 'time' keyword and value (shown in bold):

```
# ./tmp
/set type=file uid=0 gid=0 mode=0644 nlink=1
tmp              type=dir mode=01777 nlink=2 time=1089132819.370000000
# ./tmp
```

5.14.3 Storage of Specification Files

The specification files need to be accessible to mtree in order for it to be able to check the file system for changes. However, leaving the specification files on the system potentially makes them vulnerable to modification by the attacker. This can be handled as follows:

1. The specification files can be stored on a read only medium.

2. SHA-1 messages digests can be created for each file and stored offline.

The obvious choice for a read only medium would be a CD ROM. However, the process of updating the specification files would be cumbersome. An easier approach is to use a floppy disk with write-protection physically enabled on the disk. When the specification files need to be updated, the write protection can be temporarily removed. Note that physical access to the system is required to do this.

At first glance, it would appear that there is a storage capacity issue if a floppy disk is used because the size of the `usr.spec` file is approximately 2.8 Mbytes. However, it is possible to compress the specification files with `gzip` because

mtree is able to read a specification from the standard input. Thus, a check of the `/usr` hierarchy would be performed as follows:

```
zcat /floppy/usr.spec.gz | mtree -x -p /usr
```

It should be noted that leaving a floppy disk in the system will prevent the system from being remotely rebooted unless the appropriate BIOS parameter has been set to prevent the system from booting from floppy disk. Disabling the boot from floppy feature is considered to be good practice anyway and was done as part of the Honeyd system preparation.

5.14.4 Checking for Changes

OpenBSD ships with the `/etc/security` script that checks for changes to a small number of key files on a daily basis. Being alerted to the fact that a file may have changed up to 24 hours previously may be fine on some systems, but it is felt that this is too much of a delay for the Honeyd server. It would be ideal if alerting could happen in real time, but since this is not possible with mtree we will settle for running the file integrity check every hour.

The following script (`/usr/local/sbin/fic`) is used to check for changes to the files and the specification files:

```
#!/bin/sh -
#

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/sbin

umask 077

DIR=`mktemp -d /tmp/_mtree.XXXXXXXXXX` || exit 1
OUTPUT=$DIR/_mtree

trap 'rm -rf $DIR; exit 1' 0 1 2 3 13 15

mount -r /dev/fd0a /floppy/ > /dev/null 2>&1

if [ -d /floppy/mtree ] ; then
    cd /floppy/mtree
    echo "SHA1 (root.spec) = `zcat root.spec.gz | sha1`" >> $OUTPUT
    echo "SHA1 (dev.spec) = `zcat dev.spec.gz | sha1`" >> $OUTPUT
    echo "SHA1 (usr.spec) = `zcat usr.spec.gz | sha1`" >> $OUTPUT
    echo "SHA1 (usr_local.spec) = `zcat usr_local.spec.gz | sha1`" >>
$OUTPUT
    echo "SHA1 (crontab.spec) = `zcat crontab.spec.gz | sha1`" >>
$OUTPUT
    echo "Checking / ..." >> $OUTPUT
    zcat root.spec.gz | mtree -x -p / >> $OUTPUT
    echo done. >> $OUTPUT
    echo "Checking /usr ..." >> $OUTPUT
    zcat usr.spec.gz | mtree -x -p /usr >> $OUTPUT
```

```
    echo done. >> $OUTPUT
    echo "Checking /dev ..." >> $OUTPUT
    zcat dev.spec.gz | mtree -x -k nlink,size,link -p /dev >> $OUTPUT
    echo done. >> $OUTPUT
    echo "Checking /usr/local ..." >> $OUTPUT
    zcat usr_local.spec.gz | mtree -x -p /usr/local >> $OUTPUT
    echo done. >> $OUTPUT
    echo "Checking /var/cron/tabs ..." >> $OUTPUT
    zcat crontab.spec.gz | mtree -x -k nlink,size,link -p
/var/cron/tabs >> $OUTPUT
    echo done. >> $OUTPUT
else
        echo /floppy/mtree is missing >> $OUTPUT
fi

cat $OUTPUT | mailx -s "`/bin/hostname` File Integrity Check"
secops@fake.world

exit 0
```

This is run from root's crontab file and needs to be set up as follows in order to run the script every hour:

```
# File integrity checks
15  *  *  *  *  /bin/sh /usr/local/sbin/fic
```

Provided that no changes are detected, the following message will arrive at the operations console:

```
SHA1 (root.spec) = cb3efdfced56d1f5e21707e9a038437a0567edf5
SHA1 (dev.spec) = f00194d006ea3a797f3ce2f90e0c113781d4ac97
SHA1 (usr.spec) = 1bddeefa86b8651fb485942efe48de08fa70461a
SHA1 (usr_local.spec) = db76827f9d6e5c19d0e9d2f1ffb4f6de824d63e5
SHA1 (crontab.spec) = 0189ddd4ede8f32263c3b9f24a285e6a0a46600f
Checking / ...
done.
Checking /usr ...
done.
Checking /dev ...
done.
Checking /usr/local ...
done.
Checking /var/cron/tabs ...
done.
```

If the `/etc/pf.conf` file were to change, for example, the following message is generated:

```
SHA1 (root.spec) = cb3efdfced56d1f5e21707e9a038437a0567edf5
SHA1 (dev.spec) = f00194d006ea3a797f3ce2f90e0c113781d4ac97
SHA1 (usr.spec) = 1bddeefa86b8651fb485942efe48de08fa70461a
SHA1 (usr_local.spec) = db76827f9d6e5c19d0e9d2f1ffb4f6de824d63e5
SHA1 (crontab.spec) = 0189ddd4ede8f32263c3b9f24a285e6a0a46600f
```

```
Checking / ...
etc/pf.conf:
        size (2712, 2725)
        modification time (Sat Jun 26 12:53:55 2004, Wed Jul  7
17:49:48 2004)
        MD5 (5fbfda943a758aeef736287ada3ede01,
1c625eb3c93bc33191aa3cde62bc5125)
        SHA1 (e1f6aa3308b8a7bd7c31f43ad92d64fd04e9b510,
d88787abc743f181d791f2ef3091a892fe44bac8)
done.
Checking /usr ...
done.
Checking /dev ...
done.
Checking /usr/local ...
done.
Checking /var/cron/tabs ...
extra: .
done.
```

The operations console is configured to detect and alert on abnormalities in the message as well as generating an alert if the message does not arrive within 5 minutes of the expected time.

5.14.5 Putting it all together

Since the mtree specification files will need to be updated whenever there is a change to a monitored file, the steps required to perform the updates will be covered in the section 6, “Ongoing Maintenance Procedures”.

5.15 Filesystem Access Control

Objective: Implement a mechanism to prevent unauthorized changes to key files on the system in order to satisfy the requirement of RMS #17.

Time required: 10 minutes.

Although a file integrity checker has been implemented on the Honeyd server, it is preferable to try to prevent unauthorized changes from happening in the first place. One approach would be to mount file systems with the read only option. However, there are a couple of problems with this:

1. If the file system layout and placement of files has not been carefully thought out, it may not be possible to mount a particular file system as read only because it may be necessary to write to some of its directories. For example, by default NTP puts the drift file in `/etc`, which means that `/` needs to be mounted with the read and write options. However, it would

be desirable to mount the / file system as read only.

2. It is trivial to remount the file system with the read and write options if it has been mounted read only:

```
mount -uw <mount point>
```

A better approach would be to use the *system file flag*s which provide special restrictions beyond those enforced by the file permissions. Of particular interest is the *system immutable flag (schg)*, which prevents a file from being modified, moved or deleted.

5.15.1 System Security Level

The schg flag can be set at any time, but it can only be unset when the system is running at securelevel 0. In order to understand the implications of this, it is necessary to briefly discuss OpenBSD's security levels.

OpenBSD has four levels of system security:

1. **securelevel -1**. Permanently insecure mode. This means that init will not attempt to raise the security level beyond securelevel 0.

2. **securelevel 0**. Insecure mode. This mode is used during bootstrapping and while the system is in single user mode. The system flags, such as schg, can be cleared in this mode.

3. **securelevel 1**. Secure mode. This is the default security level when the system is in multi-user mode. The security level cannot be lowered while at securelevel 1 and system immutable flags cannot be removed.

4. **securelevel 2**. Highly secure mode. This includes all the features of securelevel 1 plus others. This is the recommended mode for an OpenBSD system that is functioning as a firewall because it prevents pfctl from being used to change the active rules that are being used by pf. Although the Honeyd server is not a firewall, it is desirable to prevent an attacker from being able to alter the behavior of pf. Therefore, the Honeyd server will be run at securelevel 2.

From the above, it is seen that the system immutable flags can only be removed when the system is in single user mode. In order to interact with the system in single user mode, it is necessary to be at the console because networking is not configured. This means that it should not be possible for a remote attacker to remove the schg flag in order to modify files.

5.15.2 Changing Security Levels

In securelevel 1, it is not possible to lower the security level, but it is possible to raise it. This is achieved using the `sysctl` command as shown in the following session transcript:

```
bash-2.05b$ sudo sysctl -w kern.securelevel=2
kern.securelevel: 1 -> 2
```

This change to the security level will only be effective until the next reboot because, by default OpenBSD runs at securelevel 1. In order to raise the system to securelevel 2 when the system boots, it is necessary to modify the `/etc/rc.securelevel` file as follows:

```
#securelevel=1
securelevel=2
```

As the /etc/rc.securelevel will be protected by the schg flag, it is not possible for a remote attacker to modify this file in order to set the security level to zero and then reboot the system into the insecure mode.

In order for an administrator to make changes to the system, the following command needs to be issued from the console:

```
kill -15 1
```

This will put the system in single user mode (and security level 0) where files that need changing can have the schg flag unset.

5.15.3 Protecting Files and Directories

Ideally, the schg flag should be used to protect the same files that are being checked by the file integrity checker. However, the `/dev` directory tree needs to be excluded from this list because the uid, gid and mode of some device files (e.g. the `ttyp*` files) change during the course of normal operation of the system. Making these files immutable would prevent the system from functioning properly.

The schg flag is set using the chflags command. It can set the schg flag on individual files or can recursively descend through a directory tree setting the schg on the way. In order to protect the same files that are being monitored by the file integrity checker, the following commands need to be executed:

```
chflags schg /.cshrc
chflags schg /.profile
```

```
chflags -R schg /.systrace
chflags -R schg /altroot
chflags -R schg /bin
chflags -R schg /boot
chflags schg /bsd
chflags -R schg /etc
chflags -R schg /root
chflags -R schg /sbin
chflags -R schg /usr
chflags -R schg /var/cron/tabs
```

It should be noted that “Operation not permitted” will be displayed for files that are hard links because it is not valid to set the schg flag on a file that already has the schg flag set. For example, the find command shows that the following are the same file:

```
bash-2.05b# find / -inum 204246 -print
/usr/share/zoneinfo/Etc/UTC
/usr/share/zoneinfo/Etc/Zulu
/usr/share/zoneinfo/Etc/Universal
/usr/share/zoneinfo/Zulu
/usr/share/zoneinfo/Universal
/usr/share/zoneinfo/UTC
```

It should also be noted that files should only be made immutable when no further configuration needs to be performed on the system, i.e. it should be the last task performed before the verification steps, prior to the system being put into production. Since further configuration steps will be performed in section 6, “Ongoing Maintenance Procedures”, the filesystem access control mechanism discussed here will not be enabled for the moment.

- END OF SECTION -

6 Ongoing Maintenance Procedures

Key elements to good system maintenance are:

1. Identifying *what* routine tasks need to be performed.
2. Creating procedures to define *how* to perform each task.

Where possible, these tasks should be automated with scripts in order to streamline the process and reduce the likelihood of mistakes being made. This is particularly relevant to the Honeyd server because it is intended to augment the incident response process. Not only will it be necessary for changes to be made relatively quickly, these changes will often need to be made when personnel are under pressure.

Broadly speaking the routine maintenance tasks that need to be performed on the Honeyd server are as follows:

1. Configuration changes.
2. Patching and upgrading the system.
3. Log file rotation.
4. Monitoring.
5. Updating the file integrity checker specification files.
6. Enabling and disabling filesystem access control.
7. Backups and restores.

6.1 Configuration Changes

In the course of the life of a system many configuration changes may be needed. However, focusing on the changes that are directly related to the function of the Honeyd server, the most frequent changes are likely to be to the configuration of Honeyd and pf.

Regardless of the type of change, the following general approach should be followed:

1. Disable filesystem access control.
2. Make the necessary configuration changes.
3. Update the file integrity checker specification files.
4. Enable the filesystem access control.
5. Document the changes in the Honeyd server's log book.

To put the changes described in sections 6.1.1 and 6.1.2 into context, they will occur at step 2.

It should be noted that major changes to Honeyd will require the Change Management process to be followed in order to satisfy RMS #16. This measure was designed to reduce the risk of changes to Honeyd causing disruption to the production environment. However, since the Honeyd server's role is to augment the incident response process, there is a fast track process within change management to reduce the lead times of changes that may need to be made in response to an incident.

6.1.1 Changes to Honeyd

The typical steps that will need to be followed in order to make changes to Honeyd are as follows (these steps assume that the configuration has already been verified for correction operation and that appropriate systrace policies have been generated, if necessary).

1. Kill the Honeyd process.
2. Kill the arpd process.
3. Make the required configuration changes to Honeyd by editing the `/etc/honeyd.conf` file.
4. Update the systrace policy files, if required.
5. Update the `/etc/rc.local` file with any new flags that are required by arpd or Honeyd
6. Start arpd.
7. Start honeyd.

It is felt that Honeyd was covered in sufficient detail in section 5.12 to enable the reader to perform the above steps.

6.1.2 Changes to pf

The initial configuration of pf is very general with respect to the connectivity requirements of the virtual hosts simulated by Honeyd. Therefore, it is possible that changes may need to be made to the packet filtering rules in order to accommodate more specific requirements.

Typical steps that will need to be performed are:

1. Modify the packet filtering rules by editing the `/etc/pf.conf` file.
2. Verify the syntax of the new rules.

```
sudo pfctl -n -f /etc/pf.conf
```

3. Flush the current rules and enable the new rules. It should be noted that the Honeyd server will be unprotected for a very brief period between the flushing of rules and enabling new rules.

```
sudo pfctl -Fr && pfctl -e -f /etc/pf.conf
```

Again, it is felt that pf has been covered in sufficient detail previously to enable the reader to make changes to the packet filtering rules.

6.2 Patching and Upgrading the System

The task of patching the system is most likely to apply to patching the operating system rather than to patching applications. The reason for this is simple; bugs in the applications used on the Honeyd server tend to be fixed by releasing an updated version of the application whereas bugs in the OpenBSD operating system are fixed by applying a patch to the source code and rebuilding system binaries.

6.2.1 Patching the OS

Although the procedure for patching the operating system was covered in the installation and hardening section of this document, it is important to consider the overall process for keeping the operating system up-to-date.

- **Notification**. The OpenBSD “security-announce” mailing list should be subscribed to in order to receive notification of security issues and the patches that are released to address them.

- **Obtaining**. When a new patch is released it should be obtained from the OpenBSD FTP site.

- **Testing**. Before applying patches to a production system it is wise to test them on a lab or pre-production system to ensure that no problems are introduced. OpenBSD patches are thoroughly tested before they are released, but there is no way for the developers to test for compatibility issues with every third party application that runs on the OS.

- **Backing out**. There is no official way to back out a patch, however, it is possible. The patched files can be identified by locating the corresponding *.orig files in the source tree (as discussed in section 5.5.3). The original files can be reinstated and the instructions for the particular patch followed to rebuild and reinstall the appropriate files.

- **Applying**. Provided that no issues are found in the patch testing step, the patch(es) can be applied to the production system. The high-level steps that need to be performed to apply the patches to the Honeyd server once it is production are:

    1. Remove the system immutable flag from the key system files.

    2. Transfer the source tree from the storage server to the Honeyd server.

    3. Apply the patch(es) to the source tree and rebuild the system binaries.

    4. Update the file integrity checker specification files.

    5. Remove the updated source tree from the Honeyd server and transfer to the storage server.

    6. Re-apply the system immutable flag to the system files.

    7. Perform a full system backup.

    Details for performing each of the above steps can be found elsewhere in this document.

- **Documentation**. Since there is no simple command that can be run to determine the patch level of an OpenBSD system and since the source tree will not be permanently located on the Honeyd server, it is important to keep records of all patches that have been applied to the system.

For the record, the following patches have been applied to the Honeyd server:

System: matrix				
Patch #	**Patch Type**	**Release Date**	**Architecture**	**Date Applied**
002	Security Fix	May 5, 2004	Common	June 20, 2004
003	Reliability Fix	May 5, 2004	Common	June 20, 2004
004	Reliability Fix	May 5, 2004	Common	June 20, 2004
005	Reliability Fix	May 6, 2004	Common	June 20, 2004
006	Security Fix	May 13, 2004	Common	June 20, 2004
007	Security Fix	May 20, 2004	Common	June 20, 2004
009	Security Fix	May 30, 2004	Common	June 20, 2004
010	Reliability Fix	June 9, 2004	Common	June 20, 2004
011	Security Fix	June 9, 2004	Common	June 20, 2004
012	Security Fix	June 10, 2004	Common	June 20, 2004
013	Security Fix	June 12, 2004	Common	June 20, 2004

6.2.2 Upgrading Third-Party Software

There are two types of third party software installed on the Honeyd server; software that was installed from the ports tree and software that was installed independently from source code. We need to consider how to keep up-to-date with both types of software.

In the case of software that has been installed from the ports tree, there is a tool called `out-of-date` in the `/usr/ports/infrastructure/build` directory that compares the installed packages with the versions in the ports tree. However, for this to work, two requirements need to be met:

1. The ports tree needs to be present on the system.
2. The system needs to have SSH access to the Internet to enable the latest OpenBSD 3.5 ports tree to be obtained.

As far as the Honeyd server is concerned, neither of these requirements are met. In addition, the environment in which Honeyd is deployed does not permit SSH access to the Internet. This is an issue that needs to be addressed at the policy level since there is an increasing need to be to be able to do updates to our OpenBSD systems. For the time being, updated ports trees will continue to be obtained using the author's home system, which can be used to determine if any of the packages installed on the Honeyd server need to be updated.

In the case of software that has been installed independently using source code we only need to consider arpd, Honeyd and libevent. The easiest way to keep up-to-date is to subscribe to the SecurityFocus Honeypots mailing list where Niels Provos usually announces his updates to Honeyd. It is thought that it is likely that any requirement to update arpd or libevent will be included in the announcement about Honeyd.

6.2.3 Upgrading the OS

OpenBSD is continuously being improved by its developers. Every six months a stable version of the operating system is released, which incorporates all the latest changes. Each stable release is supported for one year after which time no patches will be released to fix any security issues that are discovered. This means that we should consider upgrading the operating system on the Honeyd server every 12 months.

Since the Honeyd server is unable to use CVS, the options available to us are:

1. Upgrade from the latest CD ROM, which will be at release 3.7 by the time we need to perform this task.

2. Reinstall the OS from scratch from the latest CD ROM.

The first option can be a little messy because it will require the old configuration files to be manually merged with the new configuration files. This can be time consuming and can lead to configuration errors being introduced. In addition, stale files could be left on the system.

The preferred approach is to install a pristine system from scratch. Before embarking on this process, important data such as configuration files should be backed up.

6.3 Log file Rotation

Rotating log files is important in order to keep their size manageable so that the data that they contain is useable - there is nothing worse than trying to work with a log file that has grown to Giga Bytes.

OpenBSD uses a utility called newsyslog to rotate log files. It runs from root's crontab every hour. Rotation is performed in accordance with the directives in the `/etc/newsyslog.conf` file. The default configuration file looks like this:

```
#       $OpenBSD: newsyslog.conf,v 1.24 2003/11/11 17:00:50 jmc Exp $
#
# configuration file for newsyslog
#
# logfile_name          owner:group     mode count size when  flags
/var/cron/log           root:wheel      600  3     10   *     Z
/var/log/aculog         uucp:dialer     660  7     *    24    Z
/var/log/authlog        root:wheel      640  7     *    168   Z
/var/log/daemon                         640  5     30   *     Z
/var/log/lpd-errs                       640  7     10   *     Z
/var/log/maillog                        600  7     *    24    Z
/var/log/messages                       644  5     30   *     Z
```

```
/var/log/secure                         600  7     *    168   Z
/var/log/wtmp                           644  7     *    168   ZB
/var/log/xferlog                        640  7     250  *     Z
/var/log/ppp.log                        640  7     250  *     Z
/var/log/pflog                          600  3     250  *     ZB \
                                                  /var/run/pflogd.pid
```

The directives in the file are fairly self-evident with the key ones being *count*, *size* and *when*. *count* determines the number of generations of archived files that are kept. The existence of a value for the *size* parameter directs newsyslog to rotate the particular log file when that file exceeds the specified size (in Mbytes). The existence of a value for the *when* parameter directs newsyslog to rotate the particular log file when the number of hours since the last rotation exceeds the specified time.

Since the majority of the log data is handled by syslogd, the default values are fine because log messages are sent to three remote servers where the data is managed and retained in accordance with the organization's policy. However, as previously mentioned, pf log data is not handled by syslogd, so the default values will need to be changed in order to ensure that data is not lost before the daily backup of the `/var/log` directory occurs.

The approximate number of generations of pflog that need to be archived in a 24 hour period is calculated as follows:

- Number of IP addresses in Class A 10.0.0.0 network is 16,777,216.
- Assume that 75% of address space is unused and that Honeyd will instrument 5% of those addresses during the pilot phase of the project: 16,777,216 x 0.75 x 0.05 = 629,145 addresses.
- Assume that each of those addresses sees some probing activity on average every hour in any given 24 hour period. This gives 24 x 629,145 = 15,099,480 probes in a 24 hour period.
- Assume each probe creates three entries in pflog, each entry being approximately 132 bytes. This gives a total uncompressed data volume of 15,099,480 x 3 x 132 = 5,979,394,080 bytes or 5,702 Mbytes.
- Rotating when pflog exceeds 250 Mbytes would require around 23 log files.

Building in some margin for error, 25 would be a nice round number to use.

Nothing further needs to be done once the `newsyslog.conf` file has been modified; the next time that newsyslog runs from cron it will read the new configuration file.

6.4 Monitoring

The monitoring mechanisms have already been discussed in some detail in the installation and hardening section of this document. To recap, the following mechanisms are in place:

- Log messages are sent to remote syslog servers which have the Swatch log monitoring software running on them. Swatch has been configured to send email alerts to the operations console when an abnormal entry appears in any of the log files that are being monitored.

- An SMTP heartbeat has been implemented to check that the mechanisms used to send alerts via email are functioning. The heartbeat is sent every five minutes and if it fails to arrive at the operations console in a timely manner, a console alert will be generated.

- Output from the netstat and ps commands are piggybacked onto the SMTP heartbeat. If any deviation from the expected output is detected by the monitoring script on the operations console, an alert will be generated.

- A syslog heartbeat has been implemented to check that the mechanisms used to detect and alert on problems in the log files are functioning. The syslog heartbeat is generated every five minutes and if the corresponding email alert fails to arrive at the operations console in a timely manner, a console alert will be generated.

- A file integrity checker has been implemented to perform an hourly check for changes to important files and directories. The output of the check is sent via email to the operations console. If it fails to arrive in a timely manner or if the content of the message deviates from what is expected, an alert will be generated in the console.

- By default, OpenBSD runs daily, weekly and monthly housekeeping scripts. As well as cleaning up various temporary files, these scripts report on various aspects of the OS in order to help operations staff spot potential problems with the system.

The Operations Centre is staffed 24x7 so there are always trained personnel on hand to assess and respond to alerts.

In terms of maintenance, we need to change the Perl-based monitoring scripts on the operations console whenever changes are made that will affect the output produced in alert or heartbeat messages. Such changes are coordinated via the Change Management Process.

6.5 Updating the File Integrity Checker Specification Files

Since the mtree specification files will need to be updated whenever there is a change to a monitored file, the steps required to perform the updates are covered as part of ongoing maintenance of the Honeyd server.

It should be noted that it is very important to run the file integrity check script manually to check that no unauthorized changes have been made to the system prior to updating the specification files. If an unauthorized change is made to the system between the hourly check and the time that the specification files are regenerated, a problem could go undetected.

The procedure for updating the specification files is as follows:

1. Run the file integrity checker and examine the output.

```
/bin/sh /usr/local/sbin/fic
```

2. Unmount the floppy disk, if mounted.

```
umount /floppy
```

3. Remove write protection on the floppy disk.

4. Create a new file system on the floppy disk.

```
newfs /dev/rfd0a
```

5. Create the new specification files in `/tmp`.

```
mtree -cx -K md5digest,sha1digest -p / > /tmp/root.spec
mtree -cx -k nlink,size,link -K md5digest,sha1digest -p /dev \
> /tmp/dev.spec
mtree -cx -K md5digest,sha1digest -p /usr > /tmp/usr.spec
mtree -cx -K md5digest,sha1digest -p /usr/local \
> /tmp/usr_local.spec
mtree -cx -K md5digest,sha1digest -p /var/cron/tabs \
> /tmp/crontab.spec
```

6. Run mod-mtree-spec on `root.spec`.

```
/usr/local/sbin/mod-mtree-spec /tmp/root.spec > \
/tmp/root.spec.new
mv /tmp/root.spec.new /tmp/root.spec
```

7. Run mod-mtree-spec on `crontab.spec`.

```
/usr/local/sbin/mod-mtree-spec /tmp/crontab.spec > \
```

```
/tmp/crontab.spec.new
mv /tmp/crontab.spec.new /tmp/crontab.spec
```

8. Create SHA-1 message digests for each specification file and store them in a safe place off of the Honeyd server.

```
cd /tmp
sha1 root.spec | ssh backup@vault.fake.world \
dd of=matrix_root.spec.sha1
sha1 dev.spec | ssh backup@vault.fake.world \
dd of=matrix_dev.spec.sha1
sha1 usr.spec | ssh backup@vault.fake.world \
dd of=matrix_usr.spec.sha1
sha1 usr_local.spec | ssh backup@vault.fake.world \
dd of=matrix_usr_local.spec.sha1
sha1 crontab.spec | ssh backup@vault.fake.world \
dd of=matrix_cron.spec.sha1
```

9. Compress the specification files with gzip.

```
gzip -9 *.spec
```

10. Mount the floppy disk on `/floppy`.

```
mount /dev/fd0a /floppy
```

11. Create a directory called mtree under `/floppy`.

```
mkdir -p /floppy/mtree
```

12. Move the specification files to `/floppy/mtree`.

```
mv /tmp/*.gz /floppy/mtree/
```

13. Unmount the floppy disk.

```
umount /floppy
```

14. Enable write protection on the floppy disk.

15. Mount the floppy disk.

```
mount /dev/fd0a /floppy
```

16. Run the file integrity check shell script to verify its operation:

```
/bin/sh /usr/local/sbin/fic
```

Since there are many steps in this procedure and it will need to be performed fairly frequently, it is a natural choice for automating with a script. This script is shown in Appendix B and is called gen-spec.

6.6 Enabling and Disabling Filesystem Access Control

Before any changes can be made to the system the filesystem access control will need to be disabled by removing the system immutable flag from the appropriate files. After changes have been made, filesystem access control needs to be re-enabled before putting the system back into production.

It should be noted that while the Honeyd server has some of its defenses disabled it is good practice to protect it from being compromised by using the firewall component in the security architecture to block any traffic that is destined for it. In effect, an outer shield is put in place prior to an inner shield being lowered for maintenance.

6.6.1 Enabling File Access Control

The following procedure enables filesystem access control. It assumes that the system is in multi-user mode and at security level 1.

1. Edit `/etc/rc.securelevel` so that the system will be at security level 2 when it is booted into multi-user mode. Note that the following commands are geared towards automating the process in a script. If performing this task manually, edit the file in the normal manner.

```
cd /etc
ex rc.securelevel <<- EOF
%s/securelevel=1/securelevel=2/
w
q
EOF
```

2. Set the schg flag on the appropriate files and directories by running the following commands:

```
chflags schg /.cshrc
chflags schg /.profile
chflags -R schg /.systrace
chflags -R schg /altroot
chflags -R schg /bin
chflags -R schg /boot
chflags schg /bsd
chflags -R schg /etc
chflags -R schg /root
chflags -R schg /sbin
```

```
chflags -R schg /usr
chflags -R schg /var/cron/tabs
```

3. Raise the security level of the system from 1 to 2:

```
sysctl -w kern.securelevel=2
```

Since there are many commands in this procedure and it will need to be performed whenever a change needs to be made to the system, it is a natural choice for automating with a script. This script is shown in Appendix B and is called fac-on.

6.6.2 Disabling File Access Control

The following procedure disables the filesystem access control. It assumes that the system is in multi-user mode and at security level 2.

1. At the console, put the system into single user mode.

```
kill -15 1
```

2. Unset the schg flag on the appropriate files and directories by running some or all of the following commands as required:

```
chflags noschg /.cshrc
chflags noschg /.profile
chflags -R noschg /.systrace
chflags -R noschg /altroot
chflags -R noschg /bin
chflags -R noschg /boot
chflags noschg /bsd
chflags -R noschg /etc
chflags -R noschg /root
chflags -R noschg /sbin
chflags -R noschg /usr
chflags -R noschg /var/cron/tabs
```

3. Edit `/etc/rc.securelevel` so that the system will be at security level 1 when it is put into multi-user mode. Note that the following commands are geared towards automating the process in a script. If performing this task manually, edit the file in the normal manner.

```
cd /etc
ex rc.securelevel <<- EOF
%s/securelevel=2/securelevel=1/
w
q
```

```
EOF
```

4. Return the system to multi-user mode.

```
exit
```

Since there are many commands in this procedure and it will need to be performed whenever a change needs to be made to the system, it is a natural choice for automating with a script. This script is shown in Appendix B and is called fac-off.

6.7 Backups and Restores

A backup and restore process is being implemented in order to satisfy the requirements of RMS #23.

The purpose of the Honeyd server is to instrument unused IP address space in order to help us detect potential worm activity. Data regarding interaction with the virtual hosts that are simulated by Honeyd is sent to syslogd for logging. Since this data is valuable, we need to take steps to ensure that it is preserved. Those preservation steps include sending log messages to three separate remote syslog servers.

Unfortunately, not all logging data is handled by syslog; for example pf log data is directly logged to the `/var/log/pflog` file by pflogd., so it will be necessary to perform regular backups in order to preserve this data. Incidentally, pf log data is stored in tcpdump format which means that there is a risk of the server being compromised by tcpdump parsing a malicious payload. Therefore, it is recommended practice to move the pf log data off of a server in order to view it in a sandboxed environment.

Having investing approximately 5 hours in installing, configuring and hardening the Honeyd server, it is a good idea to perform a full system backup in case the need should arise to rebuild the system. In fact, it is a good idea to do a full system backup whenever major changes have been made to the system, e.g. after new patches have been applied and the system binaries have been rebuilt.

The backups will enable us to perform a bare-metal restore and have the system operational in a fraction of the time that it would take to rebuild it from scratch. This will enable us to quickly recover from a hardware failure or software corruption.

In the event of hardware failure it is possible to restore the system on alternate hardware because the same software is always installed on an OpenBSD system

of a particular architecture (e.g. i386) irregardless of the hardware components. However, the caveats are that the alternate system:

- Has sufficient disk capacity.
- Has no hardware compatibility issues.
- Is capable of handling the application load.

In order to provide portability between systems, backups should be performed at the filesystem (e.g. with dump or tar) level rather than at the disk level (e.g. with dd).

In the event of a compromise of the Honeyd server, the system will not be restored from backup because there is no point in putting a vulnerable system back into production. Instead the system will be "nuked from high orbit" and rebuilt from the ground up. This will include incorporating any changes required to prevent a repeat of the system compromise, which will be identified as part of the incident handling process.

In summary, the following tasks need to be addressed:

1. Full system backup.
2. Bare-metal restore.
3. Regular backup.
4. Partial backup.
5. Partial restore.

6.7.1 Full System Backup

A full system backup should be performed after the system is initially built and whenever major changes occur, such as the application of operating system patches or upgrades to third party software.

Since this particular Compaq Proliant 5500 system does not have a built-in tape drive, it will be necessary to look for an alternative backup method. My usual approach is to use SSH to transport the data to another system. Fortunately, we have a system that is used as a data repository that is backed up to tape on a regular basis. Since there is plenty of disk space on this system, it seems like an ideal place to store the backups of the Honeyd server's filesystems.

In order to perform a full system backup the server needs to be in single user mode in order to reduce system activity to a minimum. This will eliminate the risk of inconsistencies in the backup data resulting from processes writing to the filesystem while the backup is running. The backup steps will need to be performed from the console because there are no services running while the system is in single user mode. In fact, networking is not enabled by default in this mode so it will be necessary to run the `/etc/netstart` script to enable networking so that an SSH session to be established from the Honeyd server to the backup server:

```
/bin/sh /etc/netstart
```

On the Honeyd server the following command needs to be run to backup the `/` filesystem using SSH:

```
dump 0 -f - / | gzip -c -9 | ssh backup@vault.fake.world \
dd of=matrix_root.dump_`date +%F|tr -d "\012"`.gz
```

In the above example a level 0 dump (i.e. a full backup) of the `/` filesystem will be performed. Note that the data is compressed using gzip before being sent over the network in order to reduce network traffic and to reduce the disk space requirements on the destination system. The ssh command behaves like a normal pipe, so dd is used to write the bit stream to a file on the destination system (note that the data is still in dump format - dd just works at the bit level). The `date +%F|tr -d "\012"` command adds the current date (with the newline character removed) to the name of the dump file.

SSH will not be entirely transparent because it will prompt for the backup user's password. In order to eliminate the need to supply a password and pave the way to automating the backup process, we need to set up public-key authentication. The steps to do this are provided in appendix C.

Not all of the filesystems on the Honeyd server will need to be backed up; `/tmp` does not contain permanent data and `/usr/src` and `/usr/ports` are both empty at this stage. This leaves `/`, `/usr`, `/usr/local`, `/home` and `/var` that will need to be backed up.

Putting it all together, to do a full backup of the Honeyd server, the following steps need to be performed:

1. At the system console, put the Honeyd server into single user mode:

   ```
   kill -15 1
   ```

2. Restart networking on the Honeyd server:

```
/bin/sh /etc/netstart
```

3. Backup the filesystem data across the network to the destination server. The following command is used to backup the `/` filesystem:

```
dump 0 -f - / | gzip -c -9 | ssh backup@vault.fake.world \
dd of=matrix_root.dump_`date +%F|tr -d "\012"`.gz
```

4. Repeat step 3 for each filesystem on the Honeyd server that needs to be backed up. If a full backup is being performed then `/home`, `/usr`, `/usr/local`, and `/var` will also need to be backed up.

5. Put the Honeyd server back into multi-user mode by entering the `exit` command at the command prompt.

6. Record the fact that a backup has been performed in the backup log book for the Honeyd server.

It should be noted that the above approach can be used as an alternative to the method suggested in section 5.14.2 for storing the source and ports trees and moving them on and off of the system.

6.7.2 Bare-Metal Restore

The worst case scenario is a full system restore onto virgin hardware - otherwise known as a bare-metal restore. If there was a tape drive attached to the Honeyd server, this would be a simple matter of booting the system from the OpenBSD installation CD and working at the command line to partition the disks and then restore the data.

Unfortunately there is no tape drive attached, but the data is accessible remotely via the network. Therefore, we need a way to connect to the remote server from the system that needs to be restored. The approach that I found to cause least pain is to do a basic OpenBSD install on the system that needs to be restored. The installation process will only take 15 minutes and will result in a system that is partitioned in accordance with the filesystem layout specified in section 4.9, i.e. it will accomplish one of the bare-metal restore tasks. In addition, the interim OS will have networking enabled and will be able to use scp to transfer the files from the remote storage server to an unused partition on the Honeyd server. Once all the data has been transferred, the system will be booted from the OpenBSD installation CD so that data can be cleanly restored to the disk partitions.

It should be noted that this approach may not work for all systems because it relies on having a sufficiently large partition to store the compressed dump files from which a local restore will be performed. On the Honeyd server, the partition

used by the `/usr/src` filesystem can be used to temporarily hold the dump data because it is normally empty when the Honeyd server is in production.

The following steps need to be performed in order to do a full system restore:

1. Install an interim operating system from the OpenBSD 3.5 installation CD by following the five steps in section 5.3.

2. Copy the compressed dump files from the storage server to the Honeyd server (this assumes that the latest backup files are available in the backup user's home directory):

```
cd /usr/src
scp backup@vault.fake.world:./\*.gz .
```

3. Insert the OpenBSD 3.5 installation CD and reboot the system from it into the single user mode operating system.

4. Create a new filesystem on the partitions corresponding to the `/`, `/home`, `/usr`, `/usr/local` and `/var` filesystems:

```
newfs /dev/rsd0a
newfs /dev/rsd0h
newfs /dev/rsd0d
newfs /dev/rsd0g
newfs /dev/rsd0j
```

5. Mount the partition containing the compressed dump files:

```
mount /dev/sd0i /mnt2
```

6. Uncompress the dump files:

```
cd /mnt2
gzip -d *.gz
```

7. Mount the partition corresponding to the filesystem that will have its data restored. In the case below, data for the `/` filesystem will be restored so `/dev/sd0a` needs to be mounted:

```
mount /dev/sd0a /mnt
```

8. Restore the data to the filesystem:

```
cd /mnt
restore -r -f /mnt2/matrix_root.dump_*
```

9. Delete the `restoresymboltable` file:

```
rm restoresymboltable
```

10. Unmount the filesystem:

```
cd /
umount /mnt
```

11. Repeat steps 7 to 10 for the `/home`, `/usr`, `/usr/local` and `/var` filesystems.

12. Make the disk bootable and install boot blocks:

```
mount /dev/sd0a /mnt
fdisk -i sd0
cp /usr/mdec/boot /mnt/boot
/usr/mdec/installboot -v /mnt/boot /usr/mdec/biosboot sd0
```

13. Unmount the / filesystem and reboot the system:

```
umount /mnt
reboot
```

At this point the system has been restored to the state that it was in the last time a full backup was made of the system. It is possible that partial backups have been performed since that time in order to capture minor configuration changes that have been made to the system. The backup log book will indicate whether this is the case. If so, the “Partial Restores” section should be consulted.

6.7.3 Regular Backup

As previously discussed, the `/var/log` directory needs to be backed up in order to preserve the log data from pf. A daily backup should be sufficient, but if any issues arise, the frequency can be adjusted accordingly.

Since the entire contents of the directory will be backed up it makes sense to schedule it so that the daily backup captures the results of the daily, weekly and monthly maintenance scripts that run from cron. These scripts run at 01:30, 03:30 and 05:30 respectively so the backup of the /var/log directory will be scheduled to run at 07:30.

The tar command will be used to perform the backup because it backs up the files in one pass, unlike dump which does it in four passes. This will help make the backup data more reliable because the system will be active at the time that the backup is run.

The following entry needs to be added to root's crontab file in order to schedule the backup:

```
30  7  *  *  *  cd /var && tar czf - log | \
               ssh -i /root/.ssh/backup_rsa backup@vault.fake.world  \
               dd of=matrix_var_log_`date +%F|tr -d "\012"`.tgz
```

6.7.4 Partial Backup

Partial backups will occur after minor configuration changes have been made to the system. Typically, minor configuration changes will involve modification to files in `/etc` and, therefore do not warrant a full system backup.

If time permits, a partial backup of the associated filesystem can be performed using dump with the system in single user mode (as described in the section "Full System Backup"). If the priority is to get the Honeyd server up and running quickly, tar can be used to backup `/etc` using the following command:

```
cd / && tar czf - etc | ssh -i /root/.ssh/backup_rsa \
backup@vault.fake.world \
dd of=matrix_etc_`date +%F|tr -d "\012"`.tgz
```

6.7.5 Partial Restore

Partial restores are most likely to occur after a bare-metal restore has been performed or after an administration mistake has been made. The data that needs to be restored will be in dump or tar format.

Regardless of the format, we have the choice of restoring all the data or individual files. The approach taken will depend on the circumstances.

With the dump format we have the choice of restoring an entire filesystem or individual files. The steps required to restore an entire filesystem have already been described in the section on bare-metal restores. In order to restore individual files the restore command is used in interactive mode. The following session transcript shows an example of an individual file being restored:

```
bash-2.05b# cd /tmp
bash-2.05b# zcat matrix_root.dump.gz | restore -i -f -
restore > ls
.:
.cshrc     altroot/   bsd        dev/       home/      sbin/      tmp/
.profile   bin/       bsd.old    etc/       mnt/       stand/     usr/
.systrace/ boot       cdrom/     floppy/    root/      sys@       var/

restore > cd etc
```

```
restore > ls pf*
pf.conf
pf.os
restore > add pf.conf
restore > extract
set owner/mode for '.'? [yn] n
restore > q
bash-2.05b# ls
etc                 matrix_root.dump.gz
bash-2.05b# ls etc/
pf.conf
bash-2.05b#
```

After verifying that the file is correct it can be moved to `/etc`.

With the tar format the entire directory tree can be extracted to its proper location or to a temporary location such as `/tmp`. The latter case is the safest way to proceed as it allows the appropriate files to be examined before being restored to the proper location.

It should be noted that in order to perform a partial restore the appropriate system immutable flags must be disabled.

- END OF SECTION -

7 Verification of OS Configuration

The following table is an extract from the Risk Mitigation Plan (section 4.4) and shows the Risk Mitigation Steps that were implemented on the Honeyd server itself.

Reference Number	Risk Mitigation Step (RMS)	Type
4	Apply the latest OS patches.	Server
5	Run a version of Honeyd with no known vulnerabilities.	Server
6	Run Honeyd in a sandbox or jail environment.	Server
7	Implement a stack protection mechanism to help prevent buffer overflow attacks.	Server
8	Run the minimum number of services on the server.	Server
9	Use IP-based access control to limit access to services.	Server
10	Run services in a sandbox or jail environment, where possible.	Server
14	Eliminate the need for remote sessions or use encrypted sessions.	Server
17	Design and implement the file system in a manner that will make it difficult to make unauthorized system changes.	Server
18	Run file integrity checking software to detect changes to essential files such as configuration files and system binaries.	Server
20	Send logging data to a centralized syslog server to make it harder for the attacker to cover tracks.	Server
23	Implement a backup and restore process.	Server
24	Send alerts to the Operations group.	Server
25	Send regular heartbeats.	Server

The aim of this section is to verify all of the Risk Mitigation Steps listed above.

These verification steps were performed on July 21, 2004 which is referred to as "at the time of writing" from this point on. Note that the server time is set to GMT, the Honeyd server is located in the Pacific time zone so some timestamps may appear with July 22 in them.

When the verification process is complete, the Honeyd server can be put into production.

7.1 RMS #4

Objective: Verify that the latest OS patches have been applied.

Although over one month has elapsed since the “latest” patches were originally applied to the Honeyd server, at the time of writing there are no further patches that need to be added to the system.

Since patch verification was performed when the patches were applied (see section 5.5.3), no further verification is necessary. As a reminder, patch verification was performed at an early stage in the hardening process because the OpenBSD source tree was removed from the system as part of the hardening steps.

7.2 RMS #5

Objective: Verify that the version of Honeyd has no known vulnerabilities.

Checking the Honeyd website shows that at the current version is still 0.8b and recent BugTraq activity has not indicated any issues relating to Honeyd. Since known vulnerabilities are quickly fixed by Niels Provos, we can assume that there are no known vulnerabilities in version 0.8b.

Section 5.12.3 covers the verification of the basic operation of Honeyd. When Honeyd is started the resulting output confirms that version 0.8b is running on the Honeyd server.

7.3 RMS #6

Objective: Verify that Honeyd is sandboxed.

Sandboxing was implemented using systrace, which comes as part of the base OS.

In order to check that policy enforcement is working for our simple Honeyd configuration, the virtual host can be temporarily modified by adding the following line in the `/etc/honeyd.conf` file:

```
add router tcp port 80 "/usr/local/share/honeyd/scripts/web.sh"
```

After restarting Honeyd under systrace the result of trying to connect to TCP port 80 on 10.192.168.11 can be seen in the `/var/log/messages` file. An extract from this file shows that systrace denies this activity:

```
Jul 21 16:45:18 matrix systrace: deny user: nobody, prog:
/usr/local/bin/honeyd, pid: 31103(0)[1015], policy:
/usr/local/bin/honeyd, filters: 76, syscall: native-execve(59),
```

```
filename: /usr/local/share/honeyd/scripts/web.sh, argv:
/usr/local/share/honeyd/scripts/web.sh
.
.
Jul 21 16:45:18 matrix honeyd[1015]: E(10.192.250.56:1060 -
10.192.168.11:80): Operation not permitted
```

Although this is a trivial example, the appearance of “Operation not permitted” shows that systrace is only allowing Honeyd to use system calls that are present in the policy. Thus, Honeyd can be seen to be running in a sandbox.

7.4 RMS #7

Objective: Verify that the Honeyd server has a stack protection mechanism.

W^X and ProPolice are two memory protection mechanisms that were first introduced in OpenBSD 3.3. These mechanisms have been improved and are present in OpenBSD 3.5.

The operation of these mechanisms is not easily to demonstrate because it would require:

1. The Honeyd server to run a vulnerable service.
2. The development of a buffer overflow exploit for the vulnerability.
3. A deep knowledge of the operating system internals to show how W^X and ProPolice prevented the buffer overflow from occurring.

Although it is not possible to demonstrate this Risk Mitigation Step in action, OpenBSD documentation confirms that stack protection mechanisms have been implemented in OpenBSD 3.5.

7.5 RMS #8

Objective: Verify that the number of services that are running on the Honeyd server has been minimized.

This can be verified by using netstat to show the number of services listening on TCP and UDP ports and ps to show the processes running on the system.

The following output shows network services running on the Honeyd server:

```
bash-2.05b# netstat -an
Active Internet connections (including servers)
```

```
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
ip         0      0  *.*                    *.*                    255
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
tcp        0    116  10.192.168.10.22       10.192.250.64.1303
ESTABLISHED
tcp        0      0  127.0.0.1.587          *.*
LISTEN
tcp        0      0  127.0.0.1.25           *.*
LISTEN
tcp        0      0  *.22                   *.*
LISTEN
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address
(state)
udp        0      0  *.*                    *.*
udp        0      0  *.*                    *.*
udp        0      0  10.192.168.10.123      *.*
udp        0      0  127.0.0.1.123          *.*
udp        0      0  *.123                  *.*
udp        0      0  *.514                  *.*
```

Referring back to section 5.6.1, it can be seen that the services that were not required have been eliminated. The remaining services are:

- SSH (tcp 22).
- SMTP messaging (tcp 25 and tcp 587).
- NTP (udp 123) - this was added in section 5.10.
- syslog (udp 514).

There are a couple of new lines that were not present when netstat was used during one of the hardening steps to identify default services running on the Honeyd server. The following entry is created by honeyd and was verified by stopping the Honeyd process.

```
ip         0      0  *.*                    *.*                    255
```

The following entries are created by arpd and honeyd and were verified by stopping each of the processes in turn.

```
udp        0      0  *.*                    *.*
udp        0      0  *.*                    *.*
```

The following output from the ps command shows the processes running on the system.

```
bash-2.05b# ps ax
  PID TT   STAT      TIME COMMAND
```

```
    1 ??  Is      0:00.01 /sbin/init
31910 ??  Is      0:00.02 syslogd: [priv] (syslogd)
21264 ??  I       0:00.03 syslogd -a /var/empty/dev/log
 7169 ??  Is      0:00.01 pflogd: [priv] (pflogd)
20433 ??  I       0:00.02 pflogd: [running] -s 116 -f /var/log/pflog
(pflogd)
22144 ??  Is      0:00.01 /usr/sbin/sshd
23471 ??  Is      0:00.03 sendmail: accepting connections (sendmail)
 9458 ??  Is      0:00.02 /usr/local/sbin/ntpd -p /var/run/ntpd.pid -x
21584 ??  Is      0:00.00 /usr/local/sbin/arpd 10.192.168.11
 1560 ??  Is      0:00.02 /bin/systrace -a /usr/local/bin/honeyd -f
/etc/honeyd.conf -p /usr/local/share/honeyd/nmap.prints
20360 ??  Ixs     0:00.01 /usr/local/bin/honeyd -f /etc/honeyd.conf -p
/usr/local/share/honeyd/nmap.prints -x /usr/local/sh
14250 ??  Is      0:00.01 cron
26788 C0  Is+     0:00.01 /usr/libexec/getty Pc ttyC0
```

Referring back to section 5.6.4 in which the running processes were identified, it can be seen that there is now only one virtual console process running. The following processes are application processes that were added during subsequent installation and hardening steps:

- ntpd
- arpd
- systrace
- honeyd

Based on the above it can be seen that the Honeyd server is not running any unnecessary services or processes. Thus we have verified that the number of running services has been minimized.

7.6 RMS #9

Objective: Verify that pf is enforcing IP-based access control policy.

In order to verify that pf is enforcing the IP-based access control policy described in section 5.11.1, it will be necessary to send some packets to and from the Honeyd server and observe pf's response by using tcpdump to listen on the pflog0 interface.

The following session transcript shows the inbound access policy being tested:

```
bash-2.05b# tcpdump -nettti pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
Jul 21 23:46:39.982432 rule 5/0(match): pass in on fxp0:
10.192.250.64.1634 > 10.192.168.10.22: S 1351363029:1351363029(0) win
16384 <mss 1375,nop,nop,sackOK> (DF)
```

```
Jul 21 23:46:45.707092 rule 8/0(match): block in on fxp0:
10.192.168.62.59330 > 10.192.168.10.22: S 3503025196:3503025196(0) win
24820 <nop,nop,sackOK,mss 1460> (DF)
Jul 21 23:46:51.465153 rule 7/0(match): pass in on fxp0:
10.192.100.59.1761 > 10.192.168.11.23: S 1734502307:1734502307(0) win
16384 <mss 1375,nop,nop,sackOK> (DF)
Jul 21 23:47:28.769690 rule 8/0(match): block in on fxp0:
10.192.250.64.1636 > 10.192.168.10.25: S 1363625986:1363625986(0) win
16384 <mss 1375,nop,nop,sackOK> (DF)
```

Taking the output in chronological order, we see:

1. A successful connection attempt to the SSH service from one of the management VLANs.

2. An unsuccessful connection attempt to the SSH service from a system that is not on a management VLAN.

3. A packet destined for the virtual host simulated by Honeyd being allowed to pass.

4. An unsuccessful connection attempt to a service (SMTP in this case).

The above observation verifies that pf is correctly enforcing the inbound access control policy.

The following session transcript shows the outbound access policy being tested:

```
bash-2.05b# tcpdump -nettti pflog0 src 10.192.168.10
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
Jul 22 00:25:25.471719 rule 9/0(match): pass out on fxp0: 10.192.168.10
> 10.192.168.1: icmp: echo request
Jul 22 00:25:53.211911 rule 33/0(match): block out on fxp0:
10.192.168.10 > 10.192.250.1: icmp: echo request
Jul 22 00:26:23.614177 rule 10/0(match): pass out on fxp0:
10.192.168.10.40216 > 10.192.168.1.22: S 3133436122:3133436122(0) win
16384 <mss 1460,nop,nop,sackOK,[|tcp]> (DF)
Jul 22 00:26:50.823816 rule 12/0(match): pass out on fxp0:
10.192.168.10.4155 > 10.192.172.25.25: S 2254067390:2254067390(0) win
16384 <mss 1460,nop,nop,sackOK,[|tcp]> (DF)
Jul 22 00:27:41.234784 rule 33/0(match): block out on fxp0:
10.192.168.10.25547 > 10.192.118.1: S 2297057967:2297057967(0) win
16384 <mss 1460,nop,nop,sackOK,[|tcp]> (DF)
Jul 22 00:29:29.333215 rule 33/0(match): block out on fxp0:
10.192.168.10.17032 > 10.192.168.1.23: S 3344643357:3344643357(0) win
16384 <mss 1460,nop,nop,sackOK,[|tcp]> (DF)
Jul 22 00:30:06.250589 rule 33/0(match): block out on fxp0:
10.192.168.10.26014 > 10.192.168.1.53: [|domain]
Jul 22 00:31:10.907693 rule 33/0(match): block out on fxp0:
10.192.168.10.5411 > 10.192.168.1.123:  [len=5] v6 +1s server strat 101
poll 115 prec 116
```

```
Jul 22 00:032:15.779921 rule 33/0(match): block out on fxp0:
10.192.168.10.34297 > 10.192.168.1.514:  udp 5
```

Taking the output in chronological order, we see:

1. A successful ping to a device on the same subnet as the Honeyd server.
2. An unsuccessful ping attempt to a device not on the same subnet as the Honeyd server.
3. A successful connection attempt to the SSH service on another service.
4. A successful connection attempt to the SMTP service on the designated SMTP server.
5. An unsuccessful connection attempt to the SMTP service of a server that is not the designated SMTP server.
6. An unsuccessful connection attempt to a service (telnet in this case) that is not allowed to be accessed from the Honeyd server.
7. An unsuccessful attempt to send a DNS packet to a server that is not a designated DNS server.
8. An unsuccessful attempt to send an NTP packet to a server that is not a designated NTP server.
9. An unsuccessful attempt to send a syslog packet to a server that is not a designated syslog server.

It should be noted that the pf rules shown in section 5.11.1 do not log permitted packets for DNS, NTP or syslog because such entries could unnecessarily bloat the log files. Since they are not logged, they will not show up on the pflog0 device. However, as seen above, DNS, NTP and syslog packets that are not permitted to leave the Honeyd server are logged.

The above observation verifies that pf is correctly enforcing the outbound access control policy and concludes the verification of the IP-based access control mechanism.

7.7 RMS #10

Objective: Verify that services offered by the Honeyd server run in a sandbox or jail environment.

The only service offered by the Honeyd server is SSH, which is implemented using OpenSSH version 3.8.1. In OpenBSD 3.5, OpenSSH has been implemented with privilege separation[17], which confines the unprivileged child process to a chroot jail in /var/empty. Privilege separation can be seen in action when the establishment of a remote session is in progress:

```
root     23722   ??  Ss        0:00.02 sshd: smith [priv] (sshd)
sshd     20225   ??  S         0:00.04 sshd: smith [net] (sshd)
```

Note that the child process (PID 20225) is running as the unprivileged sshd user and the parent (PID 23722) is running as root.

Since the privilege separation mechanism makes it very difficult to compromise the server by running the child process in a jail environment, it satisfies the requirement of RMS #10.

7.8 RMS #14

Objective: Verify that the need for remote sessions has been eliminated or encrypted sessions are being used.

Verification of RMS #8 showed that SSH is the only remote session service active on the Honeyd server. Since SSH uses encryption, the requirement of RMS #14 is satisfied.

7.9 RMS #17

Objective: Verify that the file system has been implemented in a manner that makes it difficult to make unauthorized changes to the system.

As described in section 5.15, the Honeyd server implements a filesystem access control mechanism with the system immutable (schg) flag which protects important system configuration and executable files.

The setting of the schg flag can be verified using the ls -lo command:

```
bash-2.05b# ls -lo /sbin/
total 12912
-r-xr-xr-x  1 root  bin       schg  113324 Mar 29 11:54 ancontrol
-r-xr-xr-x  1 root  bin       schg   93196 Mar 29 11:52 atactl
-r-xr-xr-x  1 root  bin       schg   76012 Mar 29 11:52 badsect
-r-xr-xr-x  1 root  bin       schg  107660 Mar 29 11:52 brconfig
-r-xr-xr-x  1 root  bin       schg  135340 Mar 29 11:52 ccdconfig
-r-xr-xr-x  3 root  bin       schg  168684 Mar 29 11:51 chown
```

[17] http://www.citi.umich.edu/u/provos/ssh/privsep.html

.
.

To see whether it is effective we can try to replace an executable file with a trojan version:

```
bash-2.05b# mv evil_chown /sbin/chown
override r-xr-xr-x  root/bin for /sbin/chown? y
mv: rename evil_chown to /sbin/chown: Operation not permitted
```

As can be seen, this attempt was unsuccessful.

We need to check that it is not possible to unset the schg flag while the system is in multi-user mode (security level 2):

```
bash-2.05b# chflags noschg chown
chflags: chown: Operation not permitted
```

We should also make sure that it is not possible to return the system to security level 0 where it would be possible to unset the schg flag:

```
bash-2.05b# sysctl -w kern.securelevel=0
sysctl: kern.securelevel: Operation not permitted
```

An alternative method of getting the system back to a lower security level would be to check to see if the system administrator has forgotten to correctly set the securelevel value in the `/etc/rc.securelevel` file. If the value is anything other than 2, rebooting the server would put the Honeyd server into a lower security level, although it would need to be at 0 to unset the schg flag. The following session transcript shows the securelevel value in the rc.securelevel file:

```
bash-2.05b# cat rc.securelevel | grep "securelevel="
securelevel=2
```

Finally, let's check that the `/etc/rc.securelevel` file has the schg flag set so that it cannot simply be edited:

```
bash-2.05b# ls -lo rc.securelevel
-rw-r--r--  1 root  wheel schg 994 Mar 29 19:47 rc.securelevel
```

The above steps show that it would not be easy to change important files and directories in the system. Therefore, the Honeyd server satisfies the requirements of RMS #17.

7.10 RMS #18

Objective: Verify that the file integrity checker is able to detect changes to important system files.

In section 5.14.4, “Checking for Changes”, the output from the file integrity checker was shown after a change had been made to the `/etc/pf.conf` file. In this example, changes to the size, modification time and MD5 and SHA-1 hashes were detected. Thus, the Honeyd server satisfies the requirements of RMS #18.

7.11 RMS #20

Objective: Verify that log data is sent from the Honeyd server to a remote syslog server.

In order to verify this Risk Mitigation Step, it is necessary to send a message to syslogd on the Honeyd server and then check the `/var/log/messages` file on one of the remote syslog servers. The following session transcripts show this happening.

On the Honeyd server:

```
bash-2.05b$ sudo logger -p user.notice -t root remote syslog
verification
bash-2.05b$
```

On the remote syslog server:

```
-bash-2.05b$ tail -f messages
.
.
Jul 21 21:36:57 matrix.fake.world root: remote syslog verification
```

Thus it can be seen that the Honeyd server (and the supporting security architecture) satisfy the requirements of RMS #20.

7.12 RMS #23

Objective: Verify that a backup and restore process has been implemented.

All of the backup and restore processes described in Section 6.7, “Backups and Restores” are based on tried and trusted processes that we have been using in

our production environment for several years. All the procedures have been thoroughly tested to ensure that they work, including the bare-metal restore.

7.13 RMS #24

Objective: Verify that alerts are sent to the Operations Group.

A side effect of the verification of RMS #20 was that Swatch generated an alert in response to an unknown message appearing in the `/var/log/messages` file:

```
Subject: Message from Swatch
   Date: Wed, 21 Jul 2004 21:37:42 +0000 (GMT)
   From: Charlie Root <root@loghost1.fake.world>
     To: secops@fake.world

Jul 21 21:36:57 matrix.fake.world root: remote syslog verification
```

In addition, section 6.4, “Monitoring”, summarized the monitoring and alerting mechanisms that have been implemented on the Honeyd server. Examples of these alerts were included at the appropriate points within section 5, “Steps to Install and Harden the Server”.

7.14 RMS #25

Objective: Verify that regular heartbeats are sent from the Honeyd server.

The operation of the syslog and SMTP heartbeats was verified in section 5.13, “System Heartbeat”.

- END OF SECTION -

8 References

1. Oudot, Laurent. “Fighting Internet Worms with Honeypots.” Oct. 23, 2003. URL: http://www.securityfocus.com/infocus/1740 (Jul. 29, 2004).

2. Honeyd Homepage. URL: http://www.citi.umich.edu/u/provos/honeyd/ (Jul. 29, 2004).

3. The Honeynet Project Homepage. URL: http://project.honeynet.org/ (Jul. 29, 2004).

4. SecurityFocus Honeypots mailing list. URL: http://www.securityfocus.com/archive/119 (Jul. 29, 2004).

5. Spitzner, Lance. “Honeypot Definition - Almost There!” May 23, 2003. URL: http://www.securityfocus.com/archive/119/322363/2003-05-22/2003-05-28/0 (Jul. 29, 2004).

6. OpenBSD supported hardware platforms. URL: http://www.openbsd.org/plat.html (Jul. 29, 2004).

7. OpenBSD Ports. URL: http://www.openbsd.org/ports.html (Jul. 29, 2004).

8. Honeyd Remote Virtual Host Detection Vulnerability. URL: http://www.securityfocus.com/bid/9464 (Jul. 29, 2004).

9. Arpd. URL: http://www.honeyd.org/tools.php (Jul. 29, 2004).

10. OpenBSD online manual pages. URL: http://www.openbsd.org/cgi-bin/man.cgi (Jul. 29, 2004).

11. OpenBSD afterboot manual page. URL: http://www.openbsd.org/cgi-bin/man.cgi?query=afterboot (Jul. 29, 2004).

12. OpenBSD 3.5 errata and patch list. URL: http://www.openbsd.org/errata.html (Jul. 29, 2004).

13. “Applying patches in OpenBSD.” OpenBSD FAQ. URL: http://www.openbsd.org/faq/faq10.html#Patches (Jul. 29, 2004).

14. RFC 2476 - Message Submission.
URL: http://www.faqs.org/rfcs/rfc2476.html (Jul. 29, 2004).

15. Systrace Homepage.
URL: http://www.citi.umich.edu/u/provos/systrace/ (Jul. 29, 2004).

16. OpenBSD mtree manual page.
URL: http://www.openbsd.org/cgi-bin/man.cgi?query=mtree (Jul. 29, 2004).

17. Provos, Niels. “Privilege Separated OpenSSH.” Aug. 9, 2003.
URL: http://www.citi.umich.edu/u/provos/ssh/privsep.html (Jul. 29, 2004).

18. Palmer, Brandon. Nazario, Jose. “Secure Architectures with OpenBSD”. Addison-Wesley, April 2004.

19. Preston, W. Curtis. “Unix Backup & Recovery.” O’Reilly & Associates Inc., November 1999.

- END OF SECTION -

Appendices

Appendix A: Systrace Policies

Appendix B: Automation Scripts

Appendix C: Public-Key Authentication with OpenSSH

Appendix A: Systrace Policies

See section 5.12.4 for details relating to the use of these systrace policies.

Honeyd

```
Policy: /usr/local/bin/honeyd, Emulation: native
        native-issetugid: permit
        native-mprotect: permit
        native-mmap: permit
        native-__sysctl: permit
        native-fsread: filename eq "/var/run/ld.so.hints" then
permit
        native-fstat: permit
        native-close: permit
        native-fsread: filename eq "/usr/lib/libc.so.30.3" then
permit
        native-read: permit
        native-mquery: permit
        native-fsread: filename eq "/usr/lib/libpcap.so.2.1" then
permit
        native-fsread: filename eq "/usr/lib/libm.so.1.0" then
permit
        native-munmap: permit
        native-sigprocmask: permit
        native-fsread: filename eq "/etc/malloc.conf" then permit
        native-break: permit
        native-write: permit
        native-fsread: filename eq "/dev/arandom" then permit
        native-gettimeofday: permit
        native-fsread: filename eq "/usr/share/zoneinfo/GMT" then
permit
        native-getpid: permit
        native-writev: permit
        native-socket: sockdom eq "AF_UNIX" and socktype eq
"SOCK_DGRAM" then permit
        native-fcntl: permit
        native-connect: sockaddr eq "/dev/log" then permit
        native-sendto: true then permit
        native-socket: sockdom eq "AF_INET" and socktype eq
"SOCK_DGRAM" then permit
        native-fsread: filename eq
"/usr/local/share/honeyd/xprobe2.conf" then permit
        native-fsread: filename eq
"/usr/local/share/honeyd/nmap.assoc" then permit
        native-fsread: filename eq
"/usr/local/share/honeyd/nmap.prints" then permit
        native-fsread: filename eq
"/usr/local/share/honeyd/pf.os" then permit
```

```
        native-ioctl: permit
        native-fswrite: filename eq "/dev/bpf0" then permit
        native-fswrite: filename eq "/dev/bpf1" then permit
        native-fswrite: filename eq "/dev/bpf2" then permit
        native-fswrite: filename eq "/dev/bpf3" then permit
        native-fswrite: filename eq "/dev/bpf4" then permit
        native-fswrite: filename eq "/dev/bpf5" then permit
        native-fsread: filename eq "/etc/honeyd.conf" then permit
        native-fsread: filename eq "/var/run/honeyd.sock" then
permit
        native-fswrite: filename eq "/var/run/honeyd.sock" then
permit
        native-socket: sockdom eq "AF_UNIX" and socktype eq
"SOCK_STREAM" then permit
        native-setsockopt: permit
        native-bind: sockaddr eq "/var/run/honeyd.sock" then
permit
        native-listen: permit
        native-setrlimit: permit
        native-getrlimit: permit
        native-socket: sockdom eq "AF_INET" and socktype eq
"SOCK_RAW" then permit
        native-getsockopt: permit
        native-fswrite: filename eq "/var/run/honeyd.pid" then
permit
        native-fork: permit
        native-exit: permit
        native-setsid: permit
        native-fswrite: filename eq "/dev/null" then permit
        native-dup2: permit
        native-chmod: filename eq "/var/run/honeyd.pid" and mode
eq "644" then permit
        native-setgroups: permit
        native-setregid: permit
        native-setegid: gid eq "32767" then permit
        native-setgid: gid eq "32767" then permit
        native-seteuid: uid eq "32767" and uname eq "nobody" then
permit
        native-setuid: uid eq "32767" and uname eq "nobody" then
permit
        native-getgid: permit
        native-getegid: permit
        native-getuid: permit
        native-geteuid: permit
        native-setuid: uid eq "0" and uname eq "root" then permit
        native-seteuid: uid eq "0" and uname eq "root" then
permit
        native-setgid: gid eq "0" then permit
        native-setegid: gid eq "0" then permit
        native-sigaction: permit
        native-poll: permit
        native-socketpair: permit
```

```
        native-execve: filename eq
"/usr/local/share/honeyd/scripts/router-telnet.pl" and argv eq
"/usr/local/share/honeyd/scripts/router-telnet.pl" then permit
        native-sigreturn: permit
        native-wait4: permit
```

router-telnet.pl

```
Policy: /usr/local/share/honeyd/scripts/router-telnet.pl,
Emulation: native
        native-issetugid: permit
        native-mprotect: permit
        native-mmap: permit
        native-__sysctl: permit
        native-fsread: filename eq "/var/run/ld.so.hints" then
permit
        native-fstat: permit
        native-close: permit
        native-fsread: filename eq "/usr/libdata/perl5/i386-
openbsd/5.8.2/CORE" then permit
        native-fcntl: permit
        native-getdirentries: permit
        native-lseek: permit
        native-fsread: filename eq "/usr/lib/libc.so.30.3" then
permit
        native-read: permit
        native-mquery: permit
        native-fsread: filename eq "/usr/lib/libutil.so.9.0" then
permit
        native-fsread: filename eq "/usr/lib/libm.so.1.0" then
permit
        native-fsread: filename eq "/usr/lib/libperl.so.8.1" then
permit
        native-munmap: permit
        native-sigprocmask: permit
        native-fsread: filename eq "/etc/malloc.conf" then permit
        native-break: permit
        native-sigaction: permit
        native-getuid: permit
        native-geteuid: permit
        native-getgid: permit
        native-getegid: permit
        native-fsread: filename eq "/dev/arandom" then permit
        native-gettimeofday: permit
        native-fsread: filename eq
"/usr/local/share/honeyd/scripts/router-telnet.pl" then permit
        native-getpid: permit
        native-write: permit
        native-setitimer: permit
        native-exit: permit
```

Appendix B: Automation scripts

See sections 6.5 and 6.6 for details relating to these scripts.

generate-fic-spec

```
#!/bin/sh -
#
# This script should be run with the system in multi-user
# mode with filesystem access control disabled.

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/sbin

umask 077

DIR=`mktemp -d /tmp/_mtree.XXXXXXXXXX` || exit 1

trap 'rm -rf $DIR; exit 1' 0 1 2 3 13 15

echo "Running File Integrity Checker..."
/bin/sh /usr/local/sbin/fic

echo "Check that file changes are as expected before proceeding"
echo "File changes okay? (y/n): \c"
read ok
echo

if [ "$ok" = "n" ]; then
    exit 1
fi

echo "Unmounting floppy disk..."
umount /floppy

echo "Disable write protection on floppy disk"
echo "Hit enter to continue \c"
read cont
echo

echo "Creating new filesystem on floppy disk..."
newfs /dev/rfd0a

echo "Generating new mtree specification files..."
mtree -cx -K md5digest,sha1digest -p / > $DIR/root.spec
mtree -cx -k nlink,size,link -K md5digest,sha1digest -p /dev \
> $DIR/dev.spec
mtree -cx -K md5digest,sha1digest -p /usr > $DIR/usr.spec
mtree -cx -K md5digest,sha1digest -p /usr/local \
> $DIR/usr_local.spec
mtree -cx -K md5digest,sha1digest -p /var/cron/tabs \
> $DIR/crontab.spec

echo "Modifying root.spec..."
```

```
/usr/local/sbin/mod-mtree-spec $DIR/root.spec > \
$DIR/root.spec.new
mv $DIR/root.spec.new $DIR/root.spec

echo "Modifying crontab.spec..."
/usr/local/sbin/mod-mtree-spec $DIR/crontab.spec > \
$DIR/crontab.spec.new
mv $DIR/crontab.spec.new $DIR/crontab.spec

echo "Generating SHA-1 hashes and storing remotely..."
cd $DIR
sha1 root.spec | ssh -i /root/.ssh/backup_rsa backup@vault.fake.world \
dd of=matrix_root.spec.sha1
sha1 dev.spec | ssh -i /root/.ssh/backup_rsa backup@vault.fake.world \
dd of=matrix_dev.spec.sha1
sha1 usr.spec | ssh -i /root/.ssh/backup_rsa backup@vault.fake.world \
dd of=matrix_usr.spec.sha1
sha1 usr_local.spec | ssh -i /root/.ssh/backup_rsa \
backup@vault.fake.world dd of=matrix_usr_local.spec.sha1
sha1 crontab.spec | ssh -i /root/.ssh/backup_rsa \
backup@vault.fake.world dd of=matrix_cron.spec.sha1

echo "Compressing spec files..."
gzip -9 *.spec

echo "Mounting floppy disk..."
mount /dev/fd0a /floppy

mkdir -p /floppy/mtree

echo "Moving spec files to floppy disk..."
mv $DIR/*.gz /floppy/mtree/

echo "Unmounting floppy disk..."
umount /floppy

echo "Enable write protection on the floppy disk"
echo "Hit enter to continue \c"
read cont
echo

echo "Mounting floppy disk..."
mount /dev/fd0a /floppy

echo "Running File Integrity Checker..."
/bin/sh /usr/local/sbin/fic

echo "Check that no file changes are flagged"
echo "Now update the Perl monitoring script on the Ops Console"

echo "Done"

exit 0
```

fac-on

```
#!/bin/sh -
#
# This script should be run to enable the filesystem access
# control to prevent authorized changes to important system
# files.

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/sbin

umask 077

echo "WARNING: File system access control can only be disabled"
echo "from the console"
echo "Do you want to continue? (y/n): \c"
read cont
echo

if [ "$cont" != "y" ]; then
    exit 1
fi

echo "Editing /etc/rc.securelevel..."
cd /etc
ex rc.securelevel <<- EOF
%s/securelevel=1/securelevel=2/
w
q
EOF

echo "Setting the schg flag on important system files..."
chflags schg /.cshrc
chflags schg /.profile
chflags -R schg /.systrace
chflags -R schg /altroot
chflags -R schg /bin
chflags -R schg /boot
chflags schg /bsd
chflags -R schg /etc
chflags -R schg /root
chflags -R schg /sbin
chflags -R schg /usr
chflags -R schg /var/cron/tabs

echo "Raising the security level to 2..."
sysctl -w kern.securelevel=2

exit 0
```

fac-off

```
#!/bin/sh -
#
```

```
# This script should be run with the system in single user
# mode to disable filesystem access control so that
# authorized changes can be made to important system files.

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/sbin

umask 077

echo "WARNING: File system access control will be disabled"
echo "Do you want to continue? (y/n): \c"
read cont
echo

if [ "$cont" != "y" ]; then
    exit 1
fi

echo "Disabling the schg flag on important system files..."
chflags noschg /.cshrc
chflags noschg /.profile
chflags -R noschg /.systrace
chflags -R noschg /altroot
chflags -R noschg /bin
chflags -R noschg /boot
chflags noschg /bsd
chflags -R noschg /etc
chflags -R noschg /root
chflags -R noschg /sbin
chflags -R noschg /usr
chflags -R noschg /var/cron/tabs

echo "Editing /etc/rc.securelevel..."
cd /etc
ex rc.securelevel <<- EOF
%s/securelevel=2/securelevel=1/
w
q
EOF

exit 0
```

Appendix C: Public-Key Authentication with OpenSSH

Public-key authentication enables remote access to a server without the need to supply a UNIX account password. It uses two keys: a private key that is kept in a secure place on the client system and a public key that is kept on the remote server that needs to be accessed.

Ideally, the private key should be protected with a pass phrase. However, since the objective is to automate SSH access by eliminating the need to supply a password, no pass phrase will be used for the private key on the Honeyd server. This is not a big issue because, as already mentioned, access to a remote system is only possible if that system has the Honeyd server's public key on it. For this to happen, the cooperation of the remote system's administrator is required.

In order to set up public-key authentication, these steps should be followed:

1. On the client system (i.e. the Honeyd server), a private / public RSA key pair needs to be generated by the root user:

```
bash-2.05b# cd /root/.ssh
bash-2.05b# ssh-keygen -b 1024 -t rsa -f backup_rsa -N ""
Generating public/private rsa key pair.
Your identification has been saved in backup_rsa.
Your public key has been saved in backup_rsa.pub.
The key fingerprint is:
1c:99:3c:53:47:8f:5d:00:77:a8:1f:8c:04:c7:6b:4c
root@matrix.fake.world.
```

   In the session transcript above, a 1024 bit key pair has been generated and stored in the `/root/.ssh` directory. The private key file is `backup_rsa` and the public key file is `backup_rsa.pub`.

2. On the remote backup system, the system administrator needs to add the `backup_rsa.pub` key to the `~/backup/.ssh/authorized_keys` file.

3. Access to the remote server needs to be tested:

```
bash-2.05# ssh -i /root/.ssh/backup_rsa vault.fake.world
The authenticity of host 'vault.fake.world (10.192.180.50)' can't
be established.
RSA key fingerprint is
d0:d3:ed:9c:e1:4f:99:1c:a3:f0:62:35:1a:b0:ee:35
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vault.fake.world,10.192.180.50' (RSA)
to the list of known hosts.
bash-2.05#
```

4. Create a file called config in /root/.ssh and add strict host key checking to it as follows:

```
echo "StrictHostKeyChecking yes" > /root/.ssh/config
```

This will prevent the Honeyd server from being able to connect to unknown hosts.

- END OF DOCUMENT -