

Frequently Asked Questions

OpenBSD FAQ

Language: [en]

[es] [de]

[1.0 Intro to OpenBSD](#)

[2.0 Other resources](#)

[3.0 Obtaining OpenBSD](#)

[4.0 Installation Guide](#)

[5.0 Kernel Configuration](#)

[6.0 Networking Setup](#)

[7.0 Keyboard Controls](#)

[8.0 General Questions](#)

[9.0 Migrating from Linux](#)

[10.0 System Management](#)

[11.0 Performance Tuning](#)

[12.0 For Advanced Users](#)

[13.0 IPsec \[2.6\]\(#\), \[2.7\]\(#\)](#)

[14.0 Disk setup](#)

Previous releases

[2.4](#), [2.5](#)

Back to OpenBSD



This FAQ is maintained with information pertaining to the **2.6** release of **OpenBSD**. Not all information presented here may be accurate for older releases of OpenBSD. Information for previous releases is available. You should check <http://www.openbsd.com/errata.html> for important updates.

The FAQ follows release versions of OpenBSD. It will not have information pertaining to any -current options, it only attempts to track what has been released on CD. This is so there is no confusion as to which versions are being documented here.

This FAQ will take you through most critical steps to setting up your own OpenBSD system. The addressed questions range from new to advanced users. Hopefully you will find this FAQ useful. Downloadable versions of the FAQ are available in text and PDF. These versions may not be as up-to-date as the HTML versions available from this page.

- [Text Version](#)
- [PDF Version](#)

Any questions can be directed to: faq@openbsd.org

Recently updated or added FAQ's

- [6.1.1 - Identifying and Setting Up Your Network Interfaces](#) - Changed to reflect the movement of some drivers to the dc* driver.
- [6.8 - Domain Name Service - DNS, BIND, and named](#) - Helpful faq to get your DNS rolling. Submitted by Jack J. Woehr.
- [10.8 - What is the best way to add and delete users?](#) - Updated to reflect the addition of the [user\(8\)](#) utility.

The Current FAQ maintainers are:

[Eric Jackson](#)

[Wim Vandeputte](#)

[Chris Cappuccio](#)

OpenBSD FAQ Copyright © 1998-1999 OpenBSD

\$OpenBSD: index.html,v 1.95 2000/06/17 16:53:11 todd Exp \$



1.0 - Introduction to OpenBSD

Table of Contents

- [1.1 - What is OpenBSD?](#)
 - [1.2 - On what systems does OpenBSD run?](#)
 - [1.3 - Is OpenBSD really free?](#)
 - [1.4 - Why might I want to use OpenBSD?](#)
 - [1.5 - How can I help support OpenBSD?](#)
 - [1.6 - Who maintains OpenBSD?](#)
-

1.1 - What is OpenBSD?

The [OpenBSD](#) project produces a freely available, multi-platform 4.4BSD-based UNIX-like operating system. Our efforts place emphasis on [portability](#), standardization, correctness, and [security](#). [OpenBSD](#) supports binary emulation of most binaries from SVR4 (Solaris), FreeBSD, Linux, BSDI, SunOS, and HPUX.

1.2 - On what systems does OpenBSD run?

OpenBSD 2.7 runs on the following platforms:

- [i386](#) - CD bootable
- [sparc](#) - CD bootable
- [hp300](#)
- [amiga](#)
- [mac68k](#)
- [powerpc](#)
- [pmax](#)
- [mvme68k](#)

The ports for pmax, powerpc are not included on the 2.7 CDs. They can be downloaded from any of the OpenBSD ftp sites.



2.0 - Other OpenBSD information resources

Table of Contents

- [2.1 - Web Pages](#)
 - [2.2 - Mailing Lists](#)
 - [2.3 - Manual Pages](#)
 - [2.4 - Reporting Bugs](#)
-

2.1 - Web Pages of Interest

The official website for the OpenBSD project is located at: <http://www.openbsd.org/>

A lot of valuable information can be found here regarding all aspects of the OpenBSD project.

Additional information for laptop users can be found at:

<http://www.monkey.org/openbsd-mobile/>

2.2 - Mailing Lists

The OpenBSD project maintains several popular mailing lists which users should subscribe to and follow. To subscribe to a mailing list, send an e-mail message to majordomo@openbsd.org. That address is an automated subscription service. In the body of your message, on a single line, you should include a subscribe command for the list you wish to join. For example:

```
subscribe announce
```

The list processor will reply to you, asking for confirmation of your intent to join the list. The confirmation you send back to the list processor will be included in its reply to you. It will look something like this:

```
auth 90855167 subscribe announce you@example.com
```

Once you have confirmed your intent to join, you will be immediately added to the list, and the list processor will notify you that you were successfully added.

To unsubscribe from a list, you will again send an e-mail message to majordomo@openbsd.org. It might look like this:

```
unsubscribe announce
```

If you have any difficulties with the mailing list system, please first read the instructions. They can be obtained by sending an e-mail message to majordomo@openbsd.org with a message body of "help". These are the currently-available OpenBSD mailing lists:

- **announce** - Important announcements. This is a low-volume list.
- **tech** - General technical discussions
- **misc** - User questions and answers
- **bugs** - Bugs received via sendbug(1) and discussions about them.
- **source-changes** - Automated mailing of CVS source tree changes.
- **ports** - Discussion of the OpenBSD Ports Tree

- **advocacy** - Discussion of advocating OpenBSD.

Archives of the OpenBSD mailing lists can be found by visiting the mailing lists web page:

<http://www.openbsd.com/mail.html>

You can also get mail archives from <http://www.monkey.org/cgi-bin/wilma> or

<http://www.geocrawler.com/lists/4/OpenBSD>. These website contains searchable archives of the OpenBSD mailing lists.

Another mailing list that may be of interest is openbsd-mobile@monkey.org. This mailing list is a discussion of the use of OpenBSD in mobile computing.

To subscribe to this list use:

```
'echo subscribe | mail "openbsd-mobile-request@monkey.org"
```

The archives for that can be found at: <http://www.monkey.org/openbsd-mobile/archive/>

2.3 - Manual Pages

OpenBSD comes with extensive documentation in the form of manual pages, as well as longer documents relating to specific applications. To access the manual pages and other documentation, be sure that you installed the man, misc, and text distributions.

Here is a list of some of the most useful manual pages for new users:

- [afterboot\(8\)](#) - things to check after the first complete boot
- [boot\(8\)](#) - system boot strapping procedures
- [passwd.conf\(5\)](#) - format of the password configuration file
- [adduser_proc\(8\)](#) - procedure for adding new users
- [adduser\(8\)](#) - command for adding new users
- [vipw\(8\)](#) - edit the pass word file
- [man\(1\)](#) - display the on-line manual pages
- [sendbug\(1\)](#) - send a problem report (PR) about OpenBSD to a central support site.
- [disklabel\(8\)](#) - Read and write disk pack label.
- [ifconfig\(8\)](#) - configure network interface parameters.
- [route\(8\)](#) - manually manipulate the routing tables.
- [netstat\(1\)](#) - show network status.
- [reboot, halt\(8\)](#) - Stopping and restarting the system.
- [shutdown\(8\)](#) - close down the system at a given time.
- [boot_config\(8\)](#) - how to change kernel configuration at boot

Also, If you are one of the people who didn't install the man27.tgz package, you can find all the OpenBSD man pages on the web at <http://www.openbsd.org/cgi-bin/man.cgi>.

In general, if you know the name of a command or a manual page, you can read it by executing ``man command'`. For example: ``man vi'` to read about the vi editor. If you don't know the name of the command, or if ``man command'` doesn't find the manual page, you can search the manual page database by executing ``apropos something'` or ``man -k something'` where something is a likely word that might appear in the title of the manual page you're looking for. For example:

```
bsd# apropos "time zone"
tzfile (5) - time zone information
```

```
zdump (8) - time zone dumper  
zic (8) - time zone compiler
```

The parenthetical numbers indicate the section of the manual in which that page can be found. In some cases, you may find manual pages with identical names living in separate sections of the manual. For example, assume that you want to know the format of the configuration files for the cron daemon. Once you know the section of the manual for the page you want, you would execute ``man n command'` where n is the manual section number.

```
bsd# man -k cron  
cron (8) - daemon to execute scheduled commands (Vixie Cron)  
crontab (1) - maintain crontab files for individual users (V3)  
crontab (5) - tables for driving cron  
bsd# man 5 crontab
```

In addition to the UNIX manual pages, there is a typesettable document set (included in the misc distribution). It lives in the `/usr/share/doc` directory. If you also installed the text distribution, then you can format each document set with a ``make'` in the appropriate subdirectory. The psd subdirectory is the Programmer's Supplementary Documents distribution. The smm subdirectory is the System Manager's Manual. The usd subdirectory is the UNIX User's Supplementary Documents distribution. You can perform your ``make'` in the three distribution subdirectories, or you can select a specific section of a distribution and do a ``make'` in its subdirectory. Some of the subdirectories are empty. By default, formatting the documents will result in Postscript output, suitable for printing. The Postscript output can be quite large -- you should assume 250-300% increase in volume. If you do not have access to a Postscript printer or display, you may also format the documents for reading on a terminal display. In each Makefile you'll need to add the flag `-Tascii` to each instance of the `groff` commands (or execute it by hand). Some of the documents use the `ms` formatting macros, and some use the `me` macros. The Makefile in each document subdirectory (eg, `/usr/share/doc/usd/04.csh/Makefile`) will tell you which one to use. For example:

```
bsd# cd /usr/share/doc/usd/04.csh  
bsd# groff -Tascii -ms tabs csh.1 csh.2 csh.3 csh.4 csh.a csh.g > csh.txt  
bsd# more csh.txt
```

The UNIX manual pages are generally more current and trustworthy than the typesettable documents. The typesettable documents sometimes explain complicated applications in more detail than the manual pages do.

For many, having a hardcopy of the man page can be useful. Here are the guidelines to making a printable copy of a man page.

How do I display a man page source file? (i.e., one whose filename ends in a number, like `tcpdump.8`).

This is found throughout the src tree. The man pages are found in the tree unformatted, and many times through the use of [CVS](#), they will be updated. To view these pages simply :

```
# nroff -mdoc <file> | more
```

How do I get a plain man page with no formatting or control characters?

This is helpful to get the man page straight, with no non-printable characters.

Example:

```
# man <command> | col -b
```

How can I get a PostScript copy of a man page that's print-ready?

Note that `<man_src_file>` must be the man page source file (probably a file that ends in a number; i.e., `tcpdump.8`). The PostScript versions of the man pages look very nice. They can be printed or viewed on-screen with a program like `gv` (GhostView). GhostView can be found in our [Ports Tree](#).

```
# groff -mdoc -Tps <man_src_file> > outfile.ps
```

2.4 - Reporting Bugs

Before submitting any bug report, please read <http://www.openbsd.com/report.html>

Proper bug reporting is one of the most important responsibilities of end users. Very detailed information is required to diagnose most serious bugs. Developers frequently get bugs reports via e-mail such as these:

```
From: joeuser@example.com
To: bugs@openbsd.org
Subject: HELP!!!
```

I have a PC and it won't boot!!!!!! It's a 486!!!!!!

Hopefully most people understand why such reports get summarily deleted. All bug reports should contain detailed information. If Joe User had really expects someone to help find this bug, he or she will supply more information... something like this:

(Note: See [report.html](http://www.openbsd.com/report.html) for more information on creating and submitting bug reports. Basically, detailed information about your hardware is necessary if you think the bug is in any way related to your hardware or hardware configuration. Usually, dmesg output is sufficient in this respect. Next, a detailed description of your problem is necessary.)

```
From: smartuser@example.com
To: bugs@openbsd.org
Subject: 2.7 panics on an i386
```

After installing OpenBSD 2.7 from the CD which I purchased via your outstanding on-line ordering system, I find that the system halts when using any network utilities.

After booting with a bootdisk and escaping to a shell. This is the dmesg output:

```
OpenBSD 2.7 (GENERIC) #690: Fri Oct 29 16:32:17 MDT 1999
  deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
cpu0: F00F bug workaround installed
cpu0: Intel Pentium (P54C) ("GenuineIntel" 586-class) 120 MHz
cpu0: FPU,V86,DE,PSE,TSC,MSR,MCE,CX8
BIOS mem  = 654336 conventional, 15728640 extended
real mem  = 16384000
avail mem = 11112448
using 225 buffers containing 921600 bytes of memory
mainbus0 (root)
bios0 at mainbus0: AT/286+(63) BIOS, date 08/20/96
bios0: diskinfo 0xe055800c cksumlen 1 memmap 0xe0558088 apminfo 0xe0558134
apm0 at bios0: Power Management spec V1.1
apm0: battery life expectancy 95%
apm0: AC on, battery charge high, charging, estimated 1:27 minutes
pci0 at mainbus0 bus 0: configuration mode 1 (no bios)
pchb0 at pci0 dev 0 function 0 "Toshiba (2nd ID) Host-PCI" rev 0x11
"Chips and Technologies 65550" rev 0x04 at pci0 dev 4 function 0 not configured
isa0 at mainbus0
isadma0 at isa0
```

2.0 - Other OpenBSD information resources

```
wdc0 at isa0 port 0x1f0/8 irq 14
wd0 at wdc0 channel 0 drive 0: <TOSHIBA MK2720FC>
wd0: can use 16-bit, PIO mode 4
wd0: 16-sector PIO, LBA, 1296MB, 2633 cyl, 16 head, 63 sec, 2654280 sectors
sb0 at isa0 port 0x220/24 irq 5 drq 1: dsp v3.02
midi0 at sb0: <SB MIDI UART>
audio0 at sb0
opl0 at sb0: model OPL3
midil at opl0: <SB Yamaha OPL3>
wss0 at isa0 port 0x530/8 irq 10 drq 0: CS4232 (vers 63)
audiol at wss0
pcppi0 at isa0 port 0x61
midi2 at pcppi0: <PC speaker>
sysbeep0 at pcppi0
npx0 at isa0 port 0xf0/16: using exception 16
pccom0 at isa0 port 0x3f8/8 irq 4: ns16550a, 16 byte fifo
pccom1 at isa0 port 0x2f8/8 irq 3: ns16550a, 16 byte fifo
pccom2: irq 5 already in use
vt0 at isa0 port 0x60/16 irq 1: generic VGA, 80 col, color, 8 scr, mf2-kbd
pms0 at vt0 irq 12
fdc0 at isa0 port 0x3f0/6 irq 6 drq 2
fd0 at fdc0 drive 0: 1.44MB 80 cyl, 2 head, 18 sec
pcic0 at isa0 port 0x3e0/2 iomem 0xd0000/16384
pcic0 controller 0: <Intel 82365SL rev 1> has sockets A and B
pcmcia0 at pcic0 controller 0 socket 0
pcmcia1 at pcic0 controller 0 socket 1
ne3 at pcmcia1 function 0 "Linksys, EtherFast 10/100 PC Card (PCMPC100), " port
0x340/16 irq 9
ne3: address 00:e0:98:04:95:ba
pcic0: irq 11
biomask 4040 netmask 4240 ttymask 5a42
pctr: 586-class performance counters and user-level cycle counter enabled
dkcsum: wd0 matched BIOS disk 80
root on wd0a
rootdev=0x0 rrootdev=0x300 rawdev=0x302
```

Thank you!

For more information on getting the `dmesg(8)` output from a floppy, read [FAQ 14.7](#) and [FAQ 4.5](#).

If Joe User had a working OpenBSD system from which he wanted to submit a bug report, he would have used the [sendbug\(1\)](#) utility to submit his bug report to the GNATS problem tracking system. Obviously you can't use [sendbug\(1\)](#) when your system won't boot, but you should use it whenever possible. You will still need to include detailed information about what happened, the exact configuration of your system, and how to reproduce the problem. The [sendbug\(1\)](#) command requires that your system be able to deliver electronic mail successfully on the Internet.

If you have submitted a bug report and you want to check its current status without annoying anyone, the best ways are:

- Visit the GNATS tracking system: <http://cvs.openbsd.org/cgi-bin/wwwgnats.pl>.
- Look in the bugs@openbsd.org list archives: <http://www.openbsd.org/mail.html>.

[\[Back to Main Index\]](#) [\[To Section 1.0 - Introduction to OpenBSD\]](#) [\[To Section 3.0 - Obtaining OpenBSD\]](#)



www@openbsd.org

bootable means that OpenBSD will boot directly from the CD. The CD set will boot on several hardware platforms. See [section 3.0](#) of this FAQ for details of obtaining OpenBSD on CD.

Previous releases of OpenBSD also had a port for:

- [arc](#) - This port was removed from the 2.4 release. Code can be found in OpenBSD 2.3.
- [mvme88k](#)
- [alpha](#) - It is likely this port will re-appear in the near future.

Multiprocessor support? See [8.12](#) for more info.

1.3 - Is OpenBSD really free?

OpenBSD is all free. The binaries are free. The source is free. All parts of OpenBSD have reasonable copyright terms permitting free redistribution. This includes the ability to REUSE most parts of the OpenBSD source tree, either for personal or commercial purposes. OpenBSD includes NO further restrictions other than those implied by the original BSD license. Software which is written under stricter licenses cannot be included in the regular distribution of OpenBSD. This is intended to safeguard the free use of OpenBSD. For example, OpenBSD can be freely used for personal use, for academic use, by government institutions, by non-profit making organizations and by commercial organizations.

For further reading on other popular licenses read: <http://www.openbsd.com/policy.html>.

The maintainers of OpenBSD support the project largely from their own pockets. This includes the time spent programming for the project, equipment used to support the many ports, the network resources used to distribute OpenBSD to you, and the time spent answering questions and investigating users' bug reports. The OpenBSD developers are not independently wealthy and even small contributions of time, equipment, and resources make a big difference.

1.4 - Why might I want to use OpenBSD?

New users frequently want to know whether OpenBSD is superior to some other free UNIX-like operating system. That question is largely un-answerable and is the subject of countless (and useless) religious debates. Do not, under any circumstances, ask such a question on an OpenBSD mailing list.

Below are some reasons why we think OpenBSD is a useful operating system. Whether OpenBSD is right for you is a question that only you can answer.

- OpenBSD runs on many different hardware platforms.
- OpenBSD is thought of by many security professionals to be the most secure UNIX-like operating system as the result of a 10-member 1.5-year long comprehensive source code security audit.
- OpenBSD is a full-featured UNIX-like operating system available in source form at no charge.
- OpenBSD integrates cutting-edge security technology suitable for building firewalls and [private network services](#) in a distributed environment.

- OpenBSD benefits from strong on-going development in many areas, offering opportunities to work with emerging technologies with an international community of programmers and end-users.
- OpenBSD offers opportunities for ordinary people to participate in the development and testing of the product.

1.5 - How can I help support OpenBSD?

We are greatly indebted to the people and organizations that have contributed to the OpenBSD project. They are acknowledged by name here:

<http://www.openbsd.com/donations.html>

OpenBSD has a constant need for several types of support from the user community. If you find OpenBSD useful, you are strongly encouraged to find a way to contribute. If none of the suggestions below are right for you, feel free to propose an alternative by sending e-mail to

<mailto:donations@openbsd.org>

- Buy an OpenBSD CD. It includes the current full release of OpenBSD, and is bootable on many platforms. It also generates revenue to support the OpenBSD project, and reduces the strain on network resources used to deliver the distribution via the Internet. This inexpensive two-CD set includes full source. Remember, your friends need their own copy!
- Donate money. The project has a constant need for cash to pay for equipment, network connectivity, and expenses relating to CD publishing. Manufacturing CDs requires an up-front out-of-pocket investment for the OpenBSD developers, without guaranteed return. Send e-mail to donations@openbsd.org to find out how to contribute. Even small donations make a profound difference.
- Donate equipment and parts. The project has a constant need for general and specific hardware. Items such as IDE and SCSI disks, and various types of RAM are always welcome. For other types of hardware such as computer systems and motherboards, you should inquire as to current need. Write to donations@openbsd.org to arrange for shipment.
- Donate your time and skills. Programmers who enjoy writing operating systems are naturally always welcome, but there are literally dozens of other ways that people can be useful. Follow mailing lists and help answer new-user questions.
- Help maintain documentation by submitting new FAQ material (to faq@openbsd.org). Form a local user group and get your friends hooked on OpenBSD. Make a case to your employer for using OpenBSD at work. If you're a student, talk to your professors about using OpenBSD as a learning tool for Computer Science or Engineering courses. It's also worth mentioning one of the most important ways you should not try to "help" The OpenBSD project: do not waste your time engaging in operating system flame wars on Usenet newsgroups. It does not help the project to find new users and can cause substantial harm to important relationships that developers have with other developers.

1.6 - Who maintains OpenBSD?

OpenBSD is maintained by a development team spread across many different [countries](#). The project is coordinated by Theo de Raadt, located in Canada.

[\[To Section 2.0 - Other OpenBSD information resources\]](#)



www@openbsd.org

\$OpenBSD: faq1.html,v 1.19 2000/05/15 21:52:30 ericj Exp \$

3.0 - Obtaining OpenBSD

Table of Contents

- [3.1 - Buying an OpenBSD CD](#)
 - [3.1.1 - Buying OpenBSD T-Shirts](#)
 - [3.1.2 - Where can I get an OpenBSD iso image?](#)
 - [3.2 - Downloading via FTP or AFS](#)
 - [3.3 - Obtaining Current Source Code](#)
-

3.1 - Buying an OpenBSD CD

Purchasing an OpenBSD CD is generally the best way to get started. Visit the ordering page to purchase your copy: <http://www.openbsd.com/orders.html>.

There are many good reasons to own an OpenBSD CD:

- CD sales support on-going development of OpenBSD.
- Development of a multi-platform operating system requires constant investment in equipment.
- Your support in the form of a CD purchase has a real impact on future development.
- The CD contains binaries (and source) for all supported platforms.
- The CD is bootable on several platforms, and can be used to bootstrap a machine without a pre-existing installed operating system.
- The CD is useful for bootstrapping even if you choose to install a snapshot.
- Installing from CD is faster! Installing from CD preserves network connectivity resources.
- OpenBSD CD's always come with very nice stickers. Your system isn't fully complete without these. You can only get these stickers by buying a CD set or donating hardware.

If you're installing a release version of OpenBSD, you should use a CD. Snapshot releases can only be installed over the network.

3.1.1 - Buying OpenBSD T-Shirts

Yes, OpenBSD now has t-shirts for your wearing enjoyment. You can view these at <http://www.openbsd.com/tshirts.html>. Enjoy :)

3.1.2 - Does OpenBSD provide an ISO image available for download?

You can't. The official OpenBSD CD-ROM layout is copyright Theo de Raadt, as an incentive for people to buy the CD set. Note that only the layout is copyrighted, OpenBSD itself is free. Nothing precludes someone else to just grab OpenBSD and make their own CD.

3.2 - Downloading via FTP or AFS

There are numerous international mirror sites offering FTP access to OpenBSD releases and snapshots. AFS access is also available. You should always use the site closest to you. Before you begin fetching a release or snapshot, you may wish to use ping and traceroute to determine which mirror site is nearest to you and whether that nearest mirror is performing adequately. Of course, your OpenBSD release CD is always closer than any mirror. Access information is here:

<http://www.openbsd.com/ftp.html>

3.3 - Obtaining Current Source Code

Source to OpenBSD is freely redistributable and available at no charge. Generally the best way to get started with a current source tree is to install the source from the most recent CD and then configure AnonCVS to update it regularly. Information about AnonCVS, including how to set it up, is available here:

<http://www.openbsd.com/anoncvsv.html>.

or check [The OpenBSD FAQ](#)

If you don't have sufficient network bandwidth to support AnonCVS, or if your Internet access is via UUCP, you can still keep your source current by using CTM instead of AnonCVS. If that's your situation, then starting with a recent release CD is even more important. Information about CTM, including how to set it up, is available here:

<http://www.openbsd.com/ctm.html>.

Yet another alternative is to get the source code from the web. You can do that through cvsweb at:

<Http://www.openbsd.org/cgi-bin/cvsweb/>

[\[Back to Main Index\]](#) [\[To Section 2.0 - Other OpenBSD information resources\]](#) [\[To Section 4.0 - Installation Guide\]](#)



www@openbsd.org

\$OpenBSD: faq3.html,v 1.14 2000/03/25 00:05:05 wvdputte Exp \$

4.0 - Installation Guide

Table of Contents

- [4.1 - Overview of the OpenBSD Installation Procedure.](#)
 - [4.2 - What files are needed for Installation?](#)
 - [4.3 - How much space do I need for an OpenBSD installation?](#)
 - [4.4 - Multibooting OpenBSD](#)
 - [4.5 - Sending your dmesg to dmesg@openbsd.org after the install](#)
 - [My install halts at MAKEDEV all on my i386 system with 8meg of RAM.](#)
-

4.1 - Overview of the OpenBSD installation procedure

OpenBSD provides multiple ways of installing a fresh OpenBSD system for multiple architectures. This will give almost everyone in most situations the chance to install an OpenBSD system. The separate ways of installing an OpenBSD system are as follows:

- **CD-ROM** - For this you will need to have purchased an OpenBSD CD. You can do so by going to <http://www.openbsd.com/orders.html>. This is probably the easiest way of installing OpenBSD.
- **FTP** - This option requires you know your IP, Netmask, Gateway, etc. If this is so, there is an enormous list of ftp mirrors around the world which make this a speedy way of getting OpenBSD on your system.
- **DHCP** - This options is the same as **FTP** except that your IP, Netmask, Gateway and Nameservers are retrieved from the DHCP server.
- **Local Filesystem** - This option allows you to install from files on a pre-existing filesystem. Support for DOS, EXT2FS and FFS are included on the install disk.
- **NFS** - This allows you to install from files located on an nfs server. Read [mount_nfs\(8\)](#) for more information.

All these methods of installation are available on ONE floppy disk. These disks can be downloaded directly from any of the ftp mirrors. (You can get a list of mirrors on <http://www.openbsd.com/ftp.html>.) Each architecture has their own disk, so make sure you get the correct disk for your architecture. A list of supported architectures can be found in [FAQ 1](#).

Here are detailed, complete walk-through guides of the installation process for i386 and sparc architectures. Don't forget to look at the [INSTALL.i386](#) and [INSTALL.sparc](#) documents also.

- [install-i386.html](#) for **i386** architecture.
- [install-sparc.html](#) for **sparc** architecture.

People using other architectures are urged to look at the [INSTALL.arch](#) document that comes with the OpenBSD distribution for their architecture, as well as this [generic installation walkthrough](#).

Alpha architecture notes

Your alpha must use SRM firmware, as opposed to ARC. With the alpha architecture you can install by booting from the CD using CD2. You can do this by using a command such as

```
boot -fi 2.6/alpha/bsd.rd dkaX
```

Use **show device** to find your CDROM drive identifier. If you are unable to boot from the CD, you can still use the bootdisk provided at *CD2:2.6/alpha/floppy26.fs*, and write it using the ways detailed above for the i386 architecture. And again follow the [install walkthrough guide](#).

4.2 - What files are needed for Installation?

There are many packages containing the OpenBSD binaries, but which ones do you need to get your system up and running? Here is an overview of each package.

- **base26.tar.gz** - Has the base OpenBSD system **Required**
- **etc26.tar.gz** - Has all the files in /etc **Required**
- **comp26.tar.gz** - Has the compiler and its tools, libs. **Recommended**
- **man26.tar.gz** - Holds man pages **Recommended**
- **misc26.tar.gz** - Holds misc info, setup docs
- **game26.tar.gz** - Has the games for OpenBSD
- **xbase26.tar.gz** - Has the base install for X11
- **xfont26.tar.gz** - Holds X11's font server and fonts
- **xlink26.tar.gz** - Has the X servers link kit
- **xserv26.tar.gz** - Has X11's X servers
- **xshare26.tar.gz** - Has manpages, locale settings, includes, etc for X
- **bsd** - This is the Kernel. **Required**

4.3 - How much space do I need for an OpenBSD installation?

The following are suggested sub-tree sizes for a full system install. The numbers include enough extra space to permit you to run a typical home system that is connected to the internet.

- For a multiuser system, you probably want to triple these values.
- If you plan to install a significant amount of third party software, make your /usr partition large! **At least** triple these values!
- For a system that handles lots of email or web pages (stored, respectively, in /var/mail and /var/www) you will want to make your /var partition significantly larger.
- For a multiuser system which may generate lots of logs, you will still want to make your /var partition significantly larger (/var/log).

As you read this, keep in mind that /usr and /usr/X11R6 are usually both parts of the same filesystem, that is, /usr, as there is no big advantage to making them into separate filesystems.

SYSTEM	/	/usr	/var	/usr/X11R6
alpha	56M	540M	27M	161M
amiga	45M	399M	24M	36M

hp300	31M	234M	24M	47M
i386	35M	229M	24M	72M
mac68k	29M	232M	24M	36M
mvme68k	29M	232M	24M	- (no Xserver)
pmax	50M	355M	24M	60M
sparc	40M	259M	24M	49M

When you are in the disklabel editor, you may choose to make your entire system have just an 'a' (main filesystem) and 'b' (swap) . The 'a' filesystem which you set up in disklabel will become your root partition, which should be the sum of all the 3 main values above (/ , /usr, and /var) plus some space for /tmp. The 'b' partition you set up automatically becomes your system swap partition -- we recommend a minimum of 32MB but if you have disk to spare make it at least 64MB. If you have lots of disk space to spare, make this 256MB, or even 512MB.

There are four main reasons for using separate filesystems, instead of shoving everything into one or two filesystems:

- **Security:** You can mark some filesystems as 'nosuid', 'nodev', 'noexec', 'readonly', etc..
- **Stability:** A user, or a misbehaved program, can fill a filesystem with garbage, if they have write permissions for it. Your critical programs, which of course run on a different filesystem, do not get interrupted.
- **Speed:** A filesystem which gets written to frequently may get somewhat fragmented. (Luckily, the ffs filesystem, what OpenBSD uses, is not prone to heavy fragmentation.)
- **Integrity:** If one filesystem is corrupted for some reason then your other filesystems are still OK.

4.4 - Multibooting OpenBSD (i386, alpha)

OpenBSD & NT

To multiboot OpenBSD and NT, you can use NTloader, the bootloader that NT uses. To multi-boot with NT, you need a copy of your OpenBSD pbr. After running installboot, you can copy it something like this:

```
# dd if=/dev/rsd0c of=openbsd.pbr bs=512 count=1
```

Now boot NT and put openbsd.pbr in c:. Add a line like this to the end of c:\boot.ini:

```
c : \openbsd.pbr= "OpenBSD"
```

When you reboot, you should be able to select OpenBSD from the NT loader menu. There is much more information available about NTloader at the [NTLDR Hacking Guide](#).

OpenBSD & Windows or DOS (i386)

To boot OpenBSD along with Windows 3.1, Windows95, or DOS you must use a boot loader on the system that can handle OpenBSD, Windows, and DOS! Some bootloaders of choice are [osbs20b8.zip](#) or [The Ranish Partition Manager](#). Both of these are able to boot OpenBSD partitions.

OpenBSD & Linux (i386)

Please refer to [INSTALL.linux](#), which gives indepth instructions on getting OpenBSD working with Linux.

4.5 - Sending your dmesg to dmesg@openbsd.org after the install

Just to remind people, it's important for the OpenBSD developers to keep track of what hardware works, and what hardware doesn't work perfectly..

A quote from `/usr/src/etc/root/root.mail`

```
If you wish to ensure that OpenBSD runs better on your machines, please do us
a favor (after you have your mail system setup!) and type
```

```
    dmesg | mail dmesg@openbsd.org
```

```
so that we can see what kinds of configurations people are running.  We will
use this information to improve device driver support in future releases.
(We would be much happier if this information was for the supplied GENERIC
kernel; not for a custom compiled kernel).  The device driver information
we get from this helps us fix existing drivers.
```

Also check with [section 14.7](#)

Make sure you send email from an account that is able to also receive email so developers can contact you back if they have something they want you to test or change in order to get your setup working. It's not important at all to send the email from the same machine that is running OpenBSD, so if that machine is unable to receive email, just

```
    dmesg | mail your-account@yourmail.dom
```

and then forward that message to

```
    dmesg@openbsd.org
```

where `your-account@yourmail.dom` is your regular email account. (or transfer the dmesg output using `ftp/scp/floppydisk/carrier-pigeon/...`)

NOTE - Please send only GENERIC kernel dmesg's. Custom kernels that have device drivers removed are not helpful.

[\[Back to Main Index\]](#) [\[To Section 3.0 - Obtaining OpenBSD\]](#) [\[To Section 5.0 - Kernel configuration and Disk Setup\]](#)



www@openbsd.org

\$OpenBSD: faq4.html,v 1.72 2000/04/26 19:45:02 ericj Exp \$

5.0 - Kernel configuration

Table of Contents

- [5.1 - Why do I need a custom kernel?](#)
 - [5.2 - Kernel configuration Options](#)
 - [5.3 - Building your own kernel](#)
 - [5.4 - Boot-time configuration](#)
 - [5.5 - Getting more verbose output during boot](#)
 - [5.6 - Using config\(8\) to change your kernel binary](#)
-

5.1 - Why would I want to create my own custom kernel?

Several reasons, although this practice is generally geared towards knowledgeable users who have a good overall understanding of the system.

- Your computer has a very small amount of RAM and you want to preserve as much as possible by removing device drivers you don't use
- Remove default options or add options which may not have been enabled by default
- Enable experimental options

Under most circumstances you will NOT need to compile your own kernel. The GENERIC kernel will usually be all that you need. In fact, there are several reasons why you do not want to create your own kernel. The main reason is that it is very easy to make changes to the kernel configuration which look logical, but do not work. This is your danger sign. If something does not appear to work properly, please try the GENERIC kernel before sending in a bug report.

5.2 - Kernel Configuration Options

Kernel Configuration Options are options that you add to your kernel configuration that place certain features into your kernel. This allows you to have exactly the support you want, without having support for unneeded devices. There are multitudes of options that allow you to customize your kernel. Here we will go over only some of them, those that are most commonly used. Check the [options\(4\)](#) man page for a more complete list of options. You can also check the example configuration files that are available for your architecture.

Not all kernel options have been tested for compatibility with all other options. Don't put an option in your kernel unless you actually have a reason to do so! The one kernel configuration which gets the most testing is the GENERIC kernel. This is usually a combination of the options in `/usr/src/sys/arch/<your arch>/conf/GENERIC` and `/usr/src/sys/conf/GENERIC`.

- [Alpha Kernel Conf Files](#)
- [i386 Kernel Configuration files](#)
- [mac68k Kernel Configuration files](#)
- [pmax Kernel Configuration Files](#)

5.0 - Kernel configuration

- [sparc Kernel Configuration Files](#)
- [Other Arch's](#)

Look closely at these files and you will notice a line like:

```
include ".././../conf/GENERIC"
```

This means that it *is* referencing yet another configuration file. This file stores non arch-dependent options. So when creating your Kernel Config be sure to look through [/sys/conf/GENERIC](#) and see what you want. There ARE options in there that are NEEDED.

All of the options listed below should be placed in your kernel configuration file in the format of:

option OPTION

for example. To place option debug in the kernel, add a line like this:

option DEBUG

Options in the OpenBSD kernel are translated into compiler preprocessor options, therefore an option like DEBUG would have the source compiled with option -DDEBUG. Which is equivalent to doing a #define DEBUG throughout the kernel.

OpenBSD has a great many compatibility options which allow you to use binaries from other OS's. Not all are available on every architecture, so be sure to read the man pages for each option to see if your arch is supported.

- [COMPAT_SVR4\(8\)](#) - Compatibility with SVR4 binaries.
- [COMPAT_BSDOS\(8\)](#) - Compatibility with BSD/OS binaries.
- [COMPAT_LINUX\(8\)](#) - Compatibility with Linux binaries.
- [COMPAT_SUNOS\(8\)](#) - Compatibility with SunOS binaries.
- [COMPAT_ULTRIX\(8\)](#) - Compatibility with Ultrix binaries.
- [COMPAT_FREEBSD\(8\)](#) - Compatibility with FreeBSD binaries.
- COMPAT_HPUX - Compatibility with HP-UX binaries. Only available on some m68k arch's.
- [COMPAT_IBCS2\(8\)](#) - Compatibility to run ibcs2 binaries.
- COMPAT_OSF1 - Run Digital Unix binaries. Available only on Alpha platform.
- COMPAT_43 - Compatibility with 4.3BSD. Use of this is certainly discouraged, But it is needed for our Navigator port
- COMPAT_11 - Compatibility with NetBSD 1.1
- COMPAT_NOMID - Compatibility with a.out executables that lack a machine id.

It is always helpful to be able to debug problems with the kernel. But many choose not to put these options in their kernel because these options add considerable size to the kernel. They are however extremely helpful in a case where a bug might be present. This will help the developers discover the source of your problems much quicker. Here is a list of popular debugging options that can be added to your kernel.

- [DDB](#) - This compiles in the in-kernel debugger. This isn't available on all platforms. So be sure to read before adding it.
- [KGDB](#) - Compiles a remote kernel debugger using gdb's `remote target` feature.
- makeoptions DEBUG="-g" - Makes `bsd.gdb` along with `bsd`. This is useful for debugging crash dumps with `gdb`. NOTE: This option also turns on **option DEBUG**
- DEBUG - Used to put various debugging options in the kernel, where the source has defined them.
- KTRACE - Adds hooks for the system call tracing facility. Which allows users to use [ktrace\(1\)](#).
- DIAGNOSTIC - Adds code to the kernel that does internal consistency checks.
- GPROF - adds code to the kernel for kernel profiling with [kgmon\(8\)](#).
- makeoptions PROF="-pg" - The `-pg` flag causes the kernel to be compiled with support for profiling. Option GPROF is required to use this option.

Filesystem Options.

- FFS - Berkeley Fast Filesystem **NOTE:** This option is required
- EXT2FS - Second Extended File System, This is needed for those of you who want to read Linux partitions.
- MFS - Memory File System that stores files in swapable memory.
- NFS - Network File System, This is needed if you will be using NFS.
- CD9660 - This is iso9660 + rockridge filesystem. This is required to read from cd's
- MSDOSFS - Needed to read MS-DOS FAT filesystems. Also has support for Windows 95 long name + mixed case extensions.
- FDESC - Includes code for a file system which can be mounted on /dev/fd
- KERNFS - Includes code which permits the mounting of a special file system (normally mounted on /kern) in which files representing various kernel variables and parameters may be found.
- NULLFS - Code to have a loopback filesystem. [mount_null\(8\)](#) has more information
- PROCFS - Includes code for a special file system (conventionally mounted on /proc)
- PORTAL - Includes the (experimental) portal filesystem. This permits interesting tricks like opening TCP sockets by opening files in the file system.
- UMAPFS - Includes a loopback file system in which user and group ids may be remapped -- this can be useful when mounting alien file systems with different uids and gids than the local system (eg, remote NFS).
- UNION - Includes code for the union file system, which permits directories to be mounted on top of each other in such a way that both file systems remain visible. This code isn't quite stable yet.
- XFS - Add hooks for using a filesystem that is compatible with the AFS filesystem. Currently used by the Arla/AFS code.
- LFS - Log Structured Filesystem.
- FFS_SOFTUPDATES - Allows for the use of softupdates. To read up on softupdates more read the [Softupdates FAQ](#) section.
- NFSERVER - Allow for the server-side NFS code to be included in the kernel.
- NFSCLIENT - Allow for the client-side NFS code to be included in the kernel.
- FIFO - Support for FIFO's.. RECOMMENDED.
- NVNODE=*integer* - Where *integer* is the size of the cache used by the name-to-inode translation routines, (a.k.a. the namei() cache, though called by many other names in the kernel source).
- EXT2FS_SYSTEM_FLAG - This option changes the behavior of the APPEND and IMMUTABLE flags for a file on an EXT2FS filesystem. Read [options\(4\)](#) for more details.
- QUOTA - Support for Filesystem Quota's. To read up on using quotas read [FAQ 10](#)

Misc. Options

- PCIVERBOSE - Make the boot process more verbose for PCI peripherals.
- EISAVERBOSE - Make the boot process more verbose for EISA peripherals.
- PCMCIAVERBOSE - Make the boot process more verbose for PCMCIA peripherals.
- APERTURE - Provide in-kernel support for VGA framebuffer mapping by user processes. Needed to run X.
- XSERVER - Support for the X server console driver. Needed for X.
- LKM - Support for Loadable Kernel Modules. Not available on all arch's. Read [lkm\(4\)](#) for more information.
- INSECURE - Hardwires the kernel security level to -1. Read [init\(8\)](#) for more information on kernel security levels.
- MACHINE_NONCONTIG - Changes part of the VM/pmap interface, to allow for non-contiguous memory.

5.0 - Kernel configuration

- RAM_DISK_HOOKS - Allows for machine dependent functions to be called when the ramdisk driver is configured.
- RAM_DISK_IS_ROOT - Forces the ramdisk to be root.
- CCDNBUF=integer - Set number of component buffers used by CCD(4). Default is 8. For more on CCD read the [CCD\(4\)](#) man page, or the [Performance Tuning FAQ section](#).
- KMEMSTATS - This makes malloc(9), the kernel memory allocator keep statistics on its use. If option DEBUG is used, this option is automatically turned off by config.
- BOOT_CONFIG - Adds support of the -c boot option.

Networking Options

Also check the [Networking FAQ](#) or the [Networking Performance Tuning FAQ](#).

- GATEWAY - Enables IPFORWARDING and (on most ports) increases the size of NMBCLUSTERS.
- NMBCLUSTERS=integer - Controls the size mbuf cluster map.
- IPFORWARDING - Enables IP routing behavior. With this option enabled, the machine will forward IP datagrams between its interfaces that are destined for other machines.
- MROUTING - Includes support for IP multicast routers.
- INET - Includes support for the TCP/IP protocol stack. This option is REQUIRED
- MCLSHIFT=value - This option is the base-2 logarithm of the size of mbuf clusters. Read [options\(4\)](#) for more information on this option.
- NS - Include support for the Xerox XNS protocol stack. See [ns\(4\)](#)
- ISO,TPIP - Include support for the ubiquitous OSI protocol stack. See [iso\(4\)](#) for more information.
- EON - Include support for OSI tunneling over IP.
- CCITT,LLC,HDLC - Include support for the X.25 protocol stack.
- IPX, IPXIP - Include support for Internetwork Packet Exchange protocol commonly in use by Novell NetWare.
- NETATALK - Include kernel support for the AppleTalk family of protocols.
- TCP_COMPAT_42 - Use of this option is extremely discouraged, so it should not be enabled. TCP bug compatibility with 4.2BSD. In 4.2BSD, TCP sequence numbers were 32-bit signed values. Modern implementations of TCP use unsigned values.
- TCP_NEWRENO - Turns on NewReno fast recovery phase, which allows one lost segment to be recovered per round trip time.
- TCP_SACK - Turns on selective acknowledgements.
- TCP_FACK - Turns on forward acknowledgements allowing a more precise estimate of outstanding data during the fast recovery phase by using SACK information. This option can be used together with TCP_SACK.
- IPFILTER - This option enables the IP filtering on the packet level using the ipfilter package.
- IPFILTER_LOG - This option, in conjunction with IPFILTER, enables logging of IP packets using ip-filter.
- IPFILTER_DEFAULT_BLOCK - This option sets the default policy of ip-filter. If it is set, ip-filter will block packets by default.
- PPP_FILTER - This option turns on [pcap\(3\)](#) based filtering for ppp connections.
- PPP_BSDCOMP - PPP BSD compression.
- PPP_DEFLATE - Used in conjunction with PPP_BSDCOMP.
- IPSEC - This option enables IP security protocol support. See [ipsec\(4\)](#) for more details. This now implies option KEY, which gives support for PFKEYv2.
- ENCDEBUG - This option enables debugging information to be conditionally logged in case IPSEC encounters errors.
- TCP_SIGNATURE - TCP MD5 Signatures, for BGP routing sessions

PCVT Options (*only valid on i386 architecture*)

- PCVT_NScreens=integer - Number of pcvt screens. Defaults to 8.
- PCVT_VT220KEYB - If activated, a keyboard layout resembling a DEC VT200 (TM) is generated.
- PCVT_SCREENSAVER - Enables the builtin screensaver feature. Defaults to on.
- PCVT_PRETTYSCRNS - If enabled, a blinking-star screensaver is used. Default is off.
- PCVT_CTRL_ALT_DEL - If enabled, the key combination <Ctrl> <Alt> invokes a CPU reset. Default is off.
- PCVT_USEKBDSEC - Do NOT override a security lock for the keyboard. Default is on.
- PCVT_24LINESDEF - If enabled, the 25-line modi (VT emulation with 25 lines, and HP emulation with 28 lines) default to 24 lines only to provide a better compatibility to the original DEV VT220 (TM).
- PCVT_EMU_MOUSE - Emulate a three-button mouse via the keypad. Useful for notebooks when running XFree86. Defaults to off
- PCVT_META_ESC - If enabled, a sequence composed of <esc>, followed by the normal key code is emitted if a key is pressed with the <Alt> key modifier. If disabled, then normal key code with the value 0x80 added is sent. Defaults to off.

SCSI Subsystem Options

- SCsITERSE - Terser SCSI error messages. This omits the table for decoding ASC/ASCQ info, saving about 8 bytes or so.
- SCsIDEBUG - Prints extra debugging info for the SCSI subsystem to the console.

5.3 - Building your own kernel

Full instructions for creating your own custom kernel are in the [afterboot\(8\)](#) man page.

To compile your kernel from the cdrom you need to first have the source code available. You just need the kernel source to be able to compile the kernel, and this source code is available on the cd. Here is how to copy the sources from the cd. This example assumes that CD1 is mounted on /mnt.

```
# mkdir -p /usr/src/sys
# cd /mnt/sys
# tar cf - . | ( cd /usr/src/sys; tar xvf - )
```

Now to create your custom kernel it is easiest to start with the GENERIC kernel. This is located at `/usr/src/sys/arch/${arch}/conf/GENERIC`, where `${arch}` is your architecture. There are other sample configurations available in that directory as well. Here are two examples for compiling your kernel. The first example is compiling your kernel on a read-only source tree. The second on a writeable source tree.

```
# cd /somewhere
# cp /usr/src/sys/arch/$ARCH/conf/SOMEFILE .
# vi SOMEFILE (to make the changes you want)
# config -s /usr/src/sys -b . SOMEFILE
```

```
# make depend
```

(The above step is necessary when you have made any changes [including updates and patches] to your kernel source tree. This step can be ignored if you follow the next step.)

OR**# make clean**

(The above step is necessary if you make changes to your kernel configuration options, and is also recommended if you make major updates to your source tree. You can ignore this step otherwise, and just use 'make depend'.)

make

To compile a kernel inside a writeable source tree do the following:

```
# cd /sys/arch/$ARCH/conf
# vi SOMEFILE (to make any changes you want)
# config SOMEFILE (read more about it here : config\(8\))
# cd ../compile/SOMEFILE
# make
```

Where \$ARCH is the architecture you are using (e.g. i386). You can also do a **make depend** to make the dependencies for the next time you compile your kernel.

To move your kernel into place.

```
# cp /bsd /bsd.old
# cp /sys/arch/$ARCH/compile/SOMEFILE/bsd /bsd
```

To revert to your old kernel at boot you need to just

```
boot> bsd.old
```

and your old kernel will be loaded instead of /bsd.

Sometimes when you build a new kernel you will be required to install new bootblocks. To do so, read [faq14.8 on OpenBSD's Bootloader](#). Which will give you an overview on using OpenBSD's Bootloader

5.4 - Boot Time Configuration

Sometimes when booting your system you might notice that the kernel finds your device but maybe at the wrong IRQ. And maybe you need to use this device right away. Well, without rebuilding the kernel you can use OpenBSD's boot time kernel configuration. This will only correct your problem for one time. If you reboot, you will have to repeat this procedure. So, this is only meant as a temporary fix, and you should correct the problem by fixing and recompiling your kernel. Your kernel does however need **option BOOT_CONFIG** in the kernel, which GENERIC does have.

Most of this document can be found in the man page [boot_config\(8\)](#)

To boot into the User Kernel Config, or UKC, at boot time us the -c option.

```
boot> boot wd0a:/bsd -c
```

Or whichever kernel it is you want to boot. Doing this will bring up a UKC prompt. From here you can issue commands directly to the kernel specifying devices you want to change or disable or even enable.

Here is a list of common commands in the UKC.

5.0 - Kernel configuration

add **device** - Add a device through copying another

change **devno** | **device** - Modify one or more devices

disable **devno** | **device** - Disable one or more devices

enable **devno** | **device** - Enable one or more devices

find **devno** | **device** - Find one or more devices

help - Short summary of these commands

list - List ALL known devices

exit/quit - Continue Booting

show [**attr** [**val**]] - Show devices with an attribute and optional with a specified value

Once you get your device configured, use quit or exit and continue booting. After doing so you should correct your Kernel configuration and Compile a new kernel. Refer to [Building your own kernel](#) for help.

5.5 - Getting more verbose output during boot

Getting more verbose output can be very helpful when trying to debug problems when booting. If you have a problem wherein your boot floppy won't boot and need to get more information, simply reboot. When you get to the "boot>" prompt, boot with boot -c. This will bring you into the UKC>, then do:

```
UKC> verbose
autoconf verbose enabled
UKC> quit
```

Now you will be given extremely verbose output upon boot.

5.6 - Using config(8) to change your kernel

With 2.6 came the introduction of the **-e** and **-u** options with [config\(8\)](#). These options can be extremely helpful and save wasted time compiling your kernel. The **-e** flag allows you to enter the UKC or User Kernel Config on a running system. These changes will then take place on your next reboot. The **-u** flag tests to see if any changes were made to the running kernel during boot. Meaning you used **boot -c** to enter the UKC while booting your system.

The following example shows the disabling of the ep* devices from the kernel. For safety's sake you must use the **-o** option which writes the changes out to the file specified. For example : **config -e -o bsd.new /bsd** will write the changes to bsd.new. The example doesn't use the **-o** option, therefore changes are just ignored, and not written back to the kernel binary. For more information pertaining to error and warning messages read the [config\(8\)](#) man page.

```
$ sudo config -e /bsd
OpenBSD 2.6 (GENERIC) #1: Tue Nov 16 21:25:10 EST 1999
ericj@oshibana:/usr/src/sys/arch/i386/compile/GENERIC
warning: no output file specified
Enter 'help' for information
ukc> ?

      help                               Command help list
      add      dev                       Add a device
```

5.0 - Kernel configuration

base	8 10 16	Base on large numbers
change	devno dev	Change device
disable	attr val devno dev	Disable device
enable	attr val devno dev	Enable device
find	devno dev	Find device
list		List configuration
lines	count	# of lines per page
show	[attr [val]]	Show attribute
exit		Exit, without saving changes
quit		Quit, saving current changes

ukc> **list**

```

0 audio* at sb0|sb*|gus0|pas0|sp0|ess*|wss0|wss*|ym*|eap*|eso*|sv*
1 midi* at sb0|sb*|opl*|opl*|opl*|ym*|mpu*
2 nsphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
3 inphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
4 iophy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
5 expy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
6 rlphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
7 icsphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
8 sqphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
9 ukphy* at xe*|ef*|sf*|xl*|tl*|rl*|fxp* phy -1
10 scsibus* at
bt0|bt1|bt2|aha0|aha1|aha*|ahb*|ahc0|ahc*|ahc*|isp*|aic0|aic*|ncr*|adv*|adw*|sea0|uha0|uha1|uha*|wds0|atapiscsi*
11 cd* at scsibus* target -1 lun -1
12 ch* at scsibus* target -1 lun -1
13 sd* at scsibus* target -1 lun -1
14 st* at scsibus* target -1 lun -1
15 ss* at scsibus* target -1 lun -1
16 uk* at scsibus* target -1 lun -1
17 atapiscsi* at wdc0|wdc1|wdc*|wdc*|pciide* channel -1
--- more ---

```

[snip]

ukc> **disable ep**

```

31 ep0 disabled
32 ep* disabled
33 ep* disabled
87 ep0 disabled
88 ep0 disabled
89 ep* disabled
90 ep* disabled
136 ep* disabled

```

ukc> **quit**

not forced

In the above example, all ep* devices are removed from the kernel and will not be probed. In some situations where you have used the UKC during boot, via **boot -c**, you will need these changes to be written out permanently. To do this you need to use the **-u** option. In the following example, the computer was booted into the UKC and the wi(4) device was disabled. Since changes made with boot -c are NOT permanent, these changes must be written out. This example writes the changes made from boot -c into a new

5.0 - Kernel configuration

kernel binary `bsd.new`.

```
$ sudo config -e -u -o bsd.new /bsd
```

```
OpenBSD 2.6 (GENERIC) #1: Tue Nov 16 21:25:10 EST 1999
```

```
ericj@oshibana:/usr/src/sys/arch/i386/compile/GENERIC
```

```
Processing history...
```

```
151 wi* disabled
```

```
Enter 'help' for information
```

```
ukc> quit
```

[\[Back to Main Index\]](#) [\[To Section 4.0 - Installation Guide\]](#) [\[To Section 6.0 - Networking\]](#)



www@openbsd.org

\$OpenBSD: faq5.html,v 1.44 2000/05/14 21:24:25 chris Exp \$

6.0 - Networking

Table of Contents

- [6.0.1 - Before we go any further](#)
 - [6.1 - Initial network setup](#)
 - [6.2 - IP Filter](#)
 - [6.3 - Network Address Translation](#)
 - [6.4 - Dynamic Host Configuration Protocol](#)
 - [6.5 - Point to Point Protocol](#)
 - [6.6 - Tuning networking parameters](#)
 - [6.7 - Using NFS](#)
 - [6.8 - Domain Name Service - DNS, BIND, and named](#)
-

6.0.1 - Before we go any further

For the bulk of this document, it helps if you have read and at least partially understand the [Kernel Configuration and Setup](#) section of the FAQ, and the [ifconfig\(8\)](#) and [netstat\(1\)](#) man pages.

If you are a network administrator, and you are setting up routing protocols, if you are using your OpenBSD box as a router, if you need to go in depth into IP networking, you really need to read [Understanding IP addressing](#). This is an excellent document. Understanding IP addressing contains fundamental knowledge to build upon when working with IP networks!

If you are working with applications such as web servers, ftp servers, and mail servers, you may benefit greatly by [reading the RFCs](#). Of course, you can't read all of them. Rather, pick some topics that you are interested in, or that you use in your work environment. Look them up, find out how they are intended to work. The RFCs define many (thousands) of standards for protocols on the internet and how they are supposed to work.

6.1 - Initial Network Setup

6.1.1 - Identifying and Setting Up Your Network Interfaces

To start off, you must first identify your network interface. In OpenBSD, interfaces are named for the type of card, not for the type of connection. You can see your network card get initialized during boot, or after boot using the **dmesg(8)** command. You also have the ability of seeing your network interface using the **ifconfig(8)** command. For example, Here is the output in dmesg for a ne2k network card, which uses the device name ne.

```
ne3 at pcmcia1 function 0 "Linksys, EtherFast 10/100 PC Card (PCMP100), " port
0x340/16 irq 9
ne3: address 00:e0:98:04:95:ba
```

If you don't know what your device name is, here is a list of common cards and their device names.

- ne2000 Network Cards - [ne](#)
- 3Com EtherLink III and Fast EtherLink III Ethernet - [ep](#)

- 3Com EtherLink XL and Fast EtherLink XL Ethernet (3C9xx) - [xl](#)
- Intel 82586 chip Ethernet device driver - [ie](#)

Which Includes Cards Such As:

- 3Com 3C507
- AT&T StarLAN 10
- AT&T EN100
- AT&T StarLan Fiber
- Intel EtherExpress 16

- DEC/Intel 21142/3 and clone 10/100 Ethernet driver - [dc](#)

Note: Some drivers which were in OpenBSD 2.6, such as mx , ax , al and pn have been replaced by this driver.

Which Includes Cards Such As:

- Intel 21142/21143 (formerly manufactured by DEC)
- Macronix 98713, 98713A, 98715, 98715A and 98725
- Davicom DM9100 and DM9102
- ASIX Electronics AX88140A and AX88141
- ADMtek AL981 Comet and AN985 Centaur
- Lite-On 82c168 and 82c169 PNIC
- Lite-On/Macronix 82c115 PNIC II

If you are upgrading to OpenBSD 2.7 from an older release of OpenBSD, you need to pay attention here. Any references in /etc/ifaliases, /etc/ipf.rules, /etc/ipnat.rules to the old interface names mx, al, ax, or pn must be replaced with dc. Also, any hostname.xxx files with the old interface names must be renamed to hostname.dcX to be recognized. Replace the X with the interface number.

Again, you can check to see what interfaces have been identified by using the [ifconfig\(8\)](#) utility. Here is output which will show an ne2k device.

```
$ ifconfig -a
lo0: flags=8009<UP,LOOPBACK,MULTICAST>
    inet 127.0.0.1 netmask 0xff000000
lo1: flags=8008<LOOPBACK,MULTICAST>
ne3: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
    media: Ethernet manual
    inet 10.0.0.38 netmask 0xfffff00 broadcast 10.0.0.255
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST>
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST>
ppp0: flags=8010<POINTOPOINT,MULTICAST>
ppp1: flags=8010<POINTOPOINT,MULTICAST>
tun0: flags=10<POINTOPOINT>
tun1: flags=10<POINTOPOINT>
enc0: flags=0<>
bridge0: flags=0<>
bridge1: flags=0<>
```

As you can see here, [ifconfig\(8\)](#) gives us a lot more information than we are needing at this point. But it still allows us to see our interface. In the above example, the interface card is already configured. You can tell this by seeing that values are already set in "inet 10.0.0.38 netmask 0xfffff00 broadcast 10.0.0.255", and that the **UP** and **RUNNING** flags are turned on. Also, you will notice many other interfaces. Here is a list of interfaces that will be expected to be there.

- [lo](#) - Loopback Interface
- [sl](#) - SLIP Network Interface

- `ppp` - Point to Point Protocol
- `tun` - Tunnel Network Interface
- `enc` - Encapsulating Interface
- `bridge` - Ethernet bridge interface

If you don't have your interface configured the first step is to create the `/etc/hostname.${IF}` file. Where the name of your interface will take the place of `${IF}`. From the information in the examples above, the name would be `/etc/hostname.ne3`. The layout of this file is like so: To read more about the format of this file, reference the [hostname.if\(5\)](#) man page.

```
[address_family] [your_ip] [your_netmask] [media options]
```

So for the above example, a proper file would look like this:

```
$ cat /etc/hostname.ne3
inet 10.0.0.38 255.255.255.0 NONE
```

Next step from here is to setup your gateway. To do this, simply put the IP of your gateway in the file `/etc/mygate`. This will allow for your gateway to be set upon boot. From here you should setup your nameservers, and your `/etc/hosts` file. To setup your nameservers, you will create a file called `/etc/resolv.conf`. You can read more about the format of this file in the [resolv.conf\(5\)](#) man page. But for a standard usage, here is an example. In this example your domain servers are 125.2.3.4 and 125.2.3.5. You also belong in the domain "yourdomain.com".

```
$ cat /etc/resolv.conf
search yourdomain.com
nameserver 125.2.3.4
nameserver 125.2.3.5
lookup file bind
```

From here, you can either reboot or run the `/etc/netstart` script. You can do this by simply Typing (as root):

```
$ sh /etc/netstart
writing to routing socket: File exists
add net 127: gateway 127.0.0.1: File exists
writing to routing socket: File exists
add net 224.0.0.0: gateway 127.0.0.1: File exists
```

Notice that a few errors were produced, but this is in regards to the loopback interface. So this can be ignored. From here your system should be up and running. Again you can check to make sure that your interface was setup correctly with `ifconfig(8)`. You can also check your routes via `netstat(1)` or `route(8)`. Here is an example of viewing your routing tables using both.

```
$ netstat -rn
Routing tables

Internet:
Destination          Gateway              Flags          Refs          Use          Mtu  Interface
default              10.0.0.1            UGS            0             86           -    ne3
127/8                127.0.0.1          UGRS           0             0            -    lo0
127.0.0.1            127.0.0.1          UH             0             0            -    lo0
10.0.0/24            link#1              UC             0             0            -    ne3
10.0.0.1             aa:0:4:0:81:d      UHL            1             0            -    ne3
10.0.0.38            127.0.0.1          UGHS           0             0            -    lo0
224/4                127.0.0.1          URS            0             0            -    lo0

Encap:
Source              Port  Destination          Port  Proto SA(Address/SPI/Proto)
$ route show
Routing tables
```

```

Internet:
Destination      Gateway          Flags
default          10.0.0.1        UG
127.0.0.0        LOCALHOST       UG
localhost        LOCALHOST       UH
10.0.0.0          link#1          U
10.0.0.1          aa:0:4:0:81:d   UH
10.0.0.38         LOCALHOST       UGH
BASE-ADDRESS.MCA LOCALHOST        U

```

6.1.2 - Setting up your OpenBSD box as a Gateway

This is the basic information you need to set up your OpenBSD box as a gateway (also called a router). If you are using OpenBSD as a router on the Internet, we suggest that you also read the IP Filter setup instructions below to block potentially malicious traffic. Also, due to the low availability of IPv4 addresses from network service providers and regional registries, you may want to look at Network Address Translation for information on conserving your IP address space.

The GENERIC kernel is already has the ability to allow IP Forwarding, but needs to be turned on. You should do this using the **sysctl(8)** utility. To change this permanently you should edit the file `/etc/sysctl.conf` to allow for IP Forwarding. To do so add this line in that configuration file.

```
net.inet.ip.forwarding=1
```

To make this change without rebooting you would use the **sysctl(8)** utility directly. Remember though that this change will not still exist after a reboot, and needs to be run as root.

```
# sysctl -w net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
```

Now modify the routes on the other hosts on both sides. There are many possible uses of OpenBSD as a router, using software such as [routed\(8\)](#), [gated](#), [mrttd](#), and [zebra](#). OpenBSD has support in the ports collection for both [gated](#) and [mrttd](#). OpenBSD supports several T1, HSSI, ATM, FDDI, Ethernet, and serial (PPP/SLIP) interfaces.

6.1.3 - Setting up aliases on an interface.

OpenBSD has a simple mechanism for setting up ip aliases on an interface. To do this simply edit the file `/etc/ifaliases`. This file is read upon boot by the `/etc/rc` script, which is part of the [rc startup hierarchy](#). For the example, we assume that the user has an interface `dc0` and is on the network 192.168.0.0. Other important information:

- IP for `dc0` is 192.168.0.2
- NETMASK is 255.255.255.0

The format of the `/etc/ifaliases` file is:

```
[interface] [ip address] [netmask]
```

A few notes about aliases, in OpenBSD you use the interface name only. There is no differences between the first alias and the second alias. Unlike some other operating systems, OpenBSD doesn't refer to them as `dc0:0`, `dc0:1`. If you are referring to a specific aliased IP address with `ifconfig`, or adding an alias, be sure to say "`ifconfig int alias`" instead of just "`ifconfig int`" at the command line. You can delete aliases with "`ifconfig int delete`".

Assuming you are using multiple IP addresses which are in the same IP subnet with aliases, your netmask setting for each alias becomes 255.255.255.255. They do need to not follow the netmask of the first IP bound to the interface. In this example `/etc/ifaliases`, two aliases are added to the device `dc0`, which, by the way, was configured as 192.168.0.2 netmask 255.255.255.0.

```
$ cat /etc/ifaliases
dc0      192.168.0.3      255.255.255.255
```

```
dc0      192.168.0.4      255.255.255.255
```

An all ones netmask is used here so that a duplicate routing table entry for the same network is not created again.

NOTE:

Starting in OpenBSD 2.7, you can set aliases in the [/etc/hostname.*](#) file. To do the same setup as above, you would have an [/etc/hostname.dc0](#) file that is structured like this:

```
inet 192.168.0.2 255.255.255.0 media 100baseTX
inet alias 192.168.0.3 255.255.255.255 NONE
inet alias 192.168.0.4 255.255.255.255 NONE
```

Once you've made this file, it does take a reboot for it to take effect. You can, however, bring up the aliases by hand using the [ifconfig\(8\)](#) utility. To bring up the first alias you would use the command:

```
# ifconfig dc0 inet alias 192.168.0.3 netmask 255.255.255.255
```

To view these aliases you must use the command:

```
$ ifconfig -A
dc0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
      media: Ethernet manual
      inet 192.168.0.2 netmask 0xffffffff broadcast 192.168.0.255
      inet 192.168.0.3 netmask 0xffffffff broadcast 192.168.0.3
```

6.2 - IP Filter

The IP Filter package was created to handle two tasks, dealing with packet level forwarding permissions [ipf\(8\)](#) and mapping hosts/subnets to a range of external addresses [ipnat\(8\)](#). The configuration files for these two services are [/etc/ipf.rules](#) and [/etc/ipnat.rules](#).

You need to edit [/etc/rc.conf](#) to activate them at boot time. You also need to have `net.inet.ip.forwarding=1` in your [/etc/sysctl.conf](#) (or your kernel needs to have `IPFORWARDING` or `GATEWAY` options turned on.) You also need a kernel compiled with option `IPFILTER` and `IPFILTER_LOG` (the `GENERIC` kernels do have these options).

If you have IP Filter compiled into your kernel, but you don't have it turned on in your [/etc/rc.conf](#) file, you can still activate it easily.

```
# ipf -Fa -f /etc/ipf.rules -E
# ipnat -CF -f /etc/ipnat.rules
```

The `-E` flag on `ipf` 'enables' IP Filter. `-Fa` clears out any rules that you may have in there. `-f /etc/ipf.rules` loads the rules from [/etc/ipf.rules](#).

If you make changes to [/etc/ipf.rules](#) after `ipf` is loaded, you can reload your rules pretty easily!

```
# ipf -Fa -f /etc/ipf.rules
```

Same for `ipnat`..

```
# ipnat -CF -f /etc/ipnat.rules
```

This document will cover some basic `ipf` and `ipnat` configurations below. There are a lot of nice examples in [/usr/share/ipf/](#) for `ipnat` and `ipf`. We recommend you choose the one closest to what you want, and modify it to fit your needs. You can find other IP Filter information at the IP Filter [mailing list archive](#), the [IP Filter web site](#), and finally the [IP Filter HOWTO](#).

IPF

In order to enable `ipf` at boot, you will need to modify [/etc/rc.conf](#) so it reads `IPFILTER=YES`. IP Filter (`ipf`) is controlled by [/etc/ipf.rules](#), which is read at boot. For a more detailed explanation, see [ipf\(5\)](#). In the examples that follow, `fxp0` will represent the external interface to the internet. It will be different for you, based on the ethernet adaptor present in your

computer. These rules will assume full-time internet connectivity, such as you would see on a webserver.

IP Filter rules are processed sequentially from top to bottom, it helps to visualize each packet having to traverse every rule before it reaches it's destination.

For example, the default ruleset provided allows all packets to travel in, and all packets to travel out:

```
pass out from any to any
pass in from any to any
```

Now lets say we don't want to allow any incoming connections to port 3306 (mysql) because the database should only be connected to from localhost. Our ruleset would look like this:

```
pass out from any to any
pass in from any to any
block in on fxp0 from any to any port = 3306
```

This says "block in all incoming packets, from anywhere to anywhere whose destination is 3306." Essentially a packet destined for port 3306 on interface fxp0 will pass the first "pass in" rule and then be dropped by the "block in port = 3306" rule. If you reversed the order of our incoming rules (remember, order is important):

```
pass out from any to any
block in on fxp0 from any to any port = 3306
pass in from any to any
```

Packets destined for port 3306 would pass because the last rule in the set allows all packets to pass. It is important to keep this in mind when writing packet filter rules: **The last matching rule wins.**

Of course, there are exceptions to every rule. the *quick* option drops the packet at the first rule that matches. Let's look at our above flawed example, if we add *quick* to the "block in" rule:

```
pass out from any to any
block in quick on fxp0 from any to any port = 3306
pass in from any to any
```

A packet destined for our host on port 3306 will hit the "block in quick" rule and be dropped immediately. All packets destined for other ports wont find a rule match until they reach our "pass in" rule that allows all packets to pass.

Default Deny

The safest packet filtering policy is a default deny policy. All traffic not explicitly allowed is denied. This policy is far safer than explicitly denying each protected service, allows for smaller rulesets, and can protect from an accidentally misconfigured service that has been left exposed.

Let's now look at another real world example ruleset and explain things line by line. Here's an example for a webserver with a default deny policy that only allows ssh connections (for administration) and connections to http (port 80) and https (port 443).

```
#####
# begin ruleset
#####
pass in quick on fxp0 from any to any port = 22
pass in quick on fxp0 from any to any port = 80
pass in quick on fxp0 from any to any port = 443
block in quick on fxp0 from any to any
pass out on fxp0 from any to any
#####
# end ruleset
#####
```

This will allow incoming connections from anywhere to ports 22(ssh), 80(http), and 443(https). It will drop all other connection attempts, and allow all outgoing connections. This is a pretty tight ruleset. But what if you only wanted to allow internal hosts on your 1.1.1.0 address block to connect to ssh, but allow outside connections to http and https?

```
#####
# begin ruleset
#####
pass in quick on fxp0 from 1.1.1.0/24 to any port = 22
pass in quick on fxp0 from any to any port = 80
pass in quick on fxp0 from any to any port = 443
block in quick on fxp0 from any to any
pass out on fxp0 from any to any
#####
# end ruleset
#####
```

Pretty good, but what if we only want to allow one machine (1.1.1.1) to administer the web server remotely? In that case, we can change this:

```
pass in quick on fxp0 from 1.1.1.0/24 to any port = 22
```

to this:

```
pass in quick on fxp0 from 1.1.1.1/32 to any port = 22
```

IP Filter supports both CIDR and dotted decimal forms of netmask address. You could also write the above as:

```
pass in quick on fxp0 from 1.1.1.1/255.255.255.255 to any port = 22
```

but why would you?

Sample Rules

Here are some good rules for everyone to use (assuming that fxp0 is the external internet-connected interface). First we will set up simple address spoofing protection.

```
block in quick on fxp0 from 127.0.0.0/8 to any
block in quick on fxp0 from 192.168.0.0/16 to any
block in quick on fxp0 from 172.16.0.0/12 to any
block in quick on fxp0 from 10.0.0.0/8 to any
block out quick on fxp0 from any to 127.0.0.1/8
block out quick on fxp0 from any to 192.168.0.0/16
block out quick on fxp0 from any to 172.16.0.0/12
block out quick on fxp0 from any to 10.0.0.0/8
```

It's also a good idea to separate your loopback interface from your other rules.

```
pass out quick on lo0
pass in quick on lo0
```

Our ruleset is starting to look pretty good, when we put it together, here's what it looks like:

```
#####
# begin ruleset
#####
# loopback rules

pass out quick on lo0
pass in quick on lo0

# don't allow anyone to spoof non-routeable addresses

block in quick on fxp0 from 127.0.0.0/8 to any
block in quick on fxp0 from 192.168.0.0/16 to any
block in quick on fxp0 from 172.16.0.0/12 to any
block in quick on fxp0 from 10.0.0.0/8 to any
block out quick on fxp0 from any to 127.0.0.1/8
block out quick on fxp0 from any to 192.168.0.0/16
```

```

block out quick on fxp0 from any to 172.16.0.0/12
block out quick on fxp0 from any to 10.0.0.0/8

# only allow our administration machine to connect via ssh

pass in quick on fxp0 from 1.1.1.1/32 to any port = 22

# allow others to use http and https

pass in quick on fxp0 from any to any port = 80
pass in quick on fxp0 from any to any port = 443

# finally lock the rest down with a default deny

block in quick on fxp0 from any to any

# and let out-going traffic out

pass out on fxp0 from any to any

#####
# end ruleset
#####

```

Packet logging

Now that's pretty good, but it could be better. What if we want to log any connection attempts to port 22(ssh) that get blocked by our firewall? Easy, IP Filter can handle this with the *log* keyword:

```

pass in quick on fxp0 from 1.1.1.1/32 to any port = 22
block in log quick on fxp0 from any to any port = 22

```

This rule will allow our remote administration machine to connect to port 22, but deny and log all other attempts to connect to port 22.

Protocol based packet filtering

IP Filter can filter any IP protocol based on its number or name from [/etc/protocols](#). For the sake of clarity, we will only concern ourselves with tcp, udp, and icmp. These are the most commonly used protocols. All basic internet applications rely on the availability and correct operation of these protocols.

In order for ipf to filter based on protocol, the keyword *proto* must be used. To examine our earlier ssh example rule, since ssh runs over tcp, we should only allow tcp packets to connect. by using the *proto* keyword to allow only tcp, we get a rule that looks like this:

```

pass in quick on fxp0 proto tcp from 1.1.1.1/32 to any port = 22

```

But what if we need to allow connections to a service running over both tcp and udp like bind? Well, in the case of tcp/udp, IP Filter allows you to group both protocols together. Note: this only applies to tcp/udp. Using the bind example, a rule allowing tcp and udp connections in a default deny environment would look like:

```

pass in quick on fxp0 proto tcp/udp from any to any port = 53

```

Packet Filtering

In addition to filtering based on protocol, IP Filter is also capable of managing fragmented IP packets (a common method of defeating packet filters). There are two possible keywords that can be used when dealing with fragmented ip packets, *frag* for commonly fragmented IP packets, or *short* for IP packets with headers too small for comparison. Since fragmented packets can occur normally, depending on link conditions, it is best to only filter packets with headers too small for valid comparison. This can be accomplished with the following rule:

```

block in quick proto tcp all with short

```

What about IP Options? IP Filter can handle those packets too. Packets can either be dropped if they have IP options set, or they can be dropped based on the specific IP options that are set. For example, the following rule will drop and log all packets with ip options set.

```
block in log quick on fxp0 all with ipopts
```

This may however break some things such as [traceroute\(8\)](#). You can also specify which options not to allow. For instance a good rule is to block all packets with source routing options. This is accomplished with this rule:

```
block in quick on fxp0 all with opt lsrr
block in quick on fxp0 all with opt ssrr
```

TCP Flags, established connections and keeping state

Now the filtering begins in earnest. IP Filter's greatest strengths are it's ability to filter packets based on TCP flags and to maintain established connections and connection state. It is recommend that all users who wish to filter packets based on TCP flags understand what role each flag plays. For instance, if you wanted to deny all packets with the FIN, URG, and PSH flags set (like for instance an nmap OS fingerprinting attempt) you could use a rule like this:

```
block in quick on fxp0 proto tcp from any to any flags FUP
```

(Thanks to [Kyle Hargraves](#) for that tip)

IP Filter's next cool trick is it's ability to maintain state. Maintaining state has been described as "not speaking until spoken to", in other words, once a connection is established, packets no longer have to traverse rulesets. This is a very powerful feature allowing much simpler and more secure rule writing.

For example, let's see how we can apply state to our previous example ruleset (confused yet?). To review, we are allowing management access from our Class C to port 22(ssh) and allowing all incoming web traffic on ports 80(http) and 443(https). We are blocking all other traffic. But what if I want to ssh out of the webserver? What if I need to use lynx to look up something in the FAQ? Well, I can't because I have blocked all incoming connections other than on the specified ports. While this is the safest route, it can be quite inconvenient. By adding the *keep state* keywords to our "pass out" rule, we can automatically allow incoming connections in response to connections we initiate, such as when web browsing.

```
#####
# begin ruleset
#####
pass in quick on fxp0 from 1.1.1.0/24 to any port = 22
pass in quick on fxp0 from any to any port = 80
pass in quick on fxp0 from any to any port = 443
block in quick on fxp0 from any to any
pass out on fxp0 from any to any keep state
#####
# end ruleset
#####
```

This little change will dramatically increase the flexibility and security of our ruleset because IP Filter is extremely flexible. For instance, in the above ruleset, we are allowing all tcp traffic into ports 80 & 443. We can tighten this up even more. In order for a tcp connection to be established, we only need to allow the initial handshake to occur, once that occurs, we can block traffic to that port and allow our "keep state" rule to manage the connection. To allow the initial handshake to complete, we need only allow packets with the SIN and SIN/ACK flags set. By passing only packets with SIN and SIN/ACK set, we can prevent many forms of portscanning such as FIN scanning. The rules now look like this:

```
#####
# begin ruleset
#####
pass in quick on fxp0 from 1.1.1.0/24 to any port = 22
pass in quick on fxp0 from any to any port = 80 flags S/SA
pass in quick on fxp0 from any to any port = 443 flags S/SA
block in quick on fxp0 from any to any
pass out on fxp0 from any to any keep state
```

```
#####
# end ruleset
#####
```

Let's wrap this up by putting all of the rules we have so far into a ruleset. This ruleset will have a default deny policy, allow management connections from an internal network only (via ssh) and allow incoming traffic on ports 80(http) and 443(https). It will also protect against spoofed non-routeable ip addresses, and drop all packets that are too fragmented to inspect. A pretty comprehensive setup for a public webserver. Here's what /etc/ipf.rules should look like:

```
#####
# begin ruleset
#####
# loopback rules
pass out quick on lo0
pass in quick on lo0

# drop itsy bitsy frags
block in quick proto tcp all with short

# drop source routed packets
block in quick on fxp0 all with opt lsrr
block in quick on fxp0 all with opt ssrr

# don't allow anyone to spoof non-routeable addresses
block in quick on fxp0 from 127.0.0.0/8 to any
block in quick on fxp0 from 192.168.0.0/16 to any
block in quick on fxp0 from 172.16.0.0/12 to any
block in quick on fxp0 from 10.0.0.0/8 to any
block out quick on fxp0 from any to 127.0.0.1/8
block out quick on fxp0 from any to 192.168.0.0/16
block out quick on fxp0 from any to 172.16.0.0/12
block out quick on fxp0 from any to 10.0.0.0/8

# only allow our machines to connect via ssh
pass in quick on fxp0 from 1.1.1.0/24 to any port = 22

# allow others to use http and https
pass in quick on fxp0 from any to any port = 80 flags S/SA
pass in quick on fxp0 from any to any port = 443 flags S/SA

# finally lock the rest down with a default deny
block in quick on fxp0 from any to any

# and let out-going traffic out and maintain state on established connections
pass out on fxp0 from any to any keep state

#####
# end ruleset
#####
```

For further information on ipf, the [IPF how-to](#) is an excellent source, as are the resources available at the [IP Filter](#) homepage.

6.3 - IPNAT

Initial work done by Wayne Fergerstrom <wayne@methadonia.net>

6.3.1 NAT Introduction

Section Introduction

This section attempts to aid for those installing and configuring Network Address Translation ("NAT") on an OpenBSD machine. The user is assumed to have already set up and configured an OpenBSD machine with two network cards (one connected to the Internet and the other to the local network). IP Network Address Translation will work on machines with only one NIC, however since packets will be going in and out of the same interface, ethernet collisions will slow down performance considerably.

Based on [RFC 1631](#), ipnat provides an easy way to map internal networks to a single routeable ("real") internet address. This is very useful if you don't have officially assigned addresses for every host on your internal network. When you set up private/internal networks, you can take advantage of reserved address blocks (assigned in [RFC 1918](#)), such as:

10.0.0.0/8 (10.0.0.0 - 10.255.255.255)
 172.16.0.0/12 (172.16.0.0 - 172.31.255.255)
 192.168.0.0/16 (192.168.0.0 - 192.168.255.255)

Terminology

The conventions used in this document are fairly straightforward. For documentation purposes I will review some of the terms and format for which this document adheres to.

"NAT"

This describes the function of "Network Address Translation." The process of NAT is described later in this document.

"ipnat"

This is short for "IP Network Address Translation." In-and-of itself, it can be used interchangeably with NAT. However, in this document the term "ipnat" will be used solely for command-line only use.

"IPF"

This is short for "IP Filter." IP Filter is a portable packet filtering software that is included as part of OpenBSD. IP Filter must be enabled before you can turn on ipnat. This is easy, just edit /etc/rc.conf and change ipf=NO to ipf=YES. That only changes it for the boot up sequence, you also need to do 'ipf -E' to turn on ipf while you are booted. Of course, this is described further, below.

Configuration

This is how the computers are setup concerning this document. Your setup will vary from this, but the purpose of the document is to give you an overview so you can conform this information to your setup.

Computer Operating System: OpenBSD v2.7 i386

NICs:

NetGear 10/100MB **dc0**
 Connected to the EXTERNAL LAN (or WAN)
IP Address: 24.5.0.5
Netmask: 255.255.255.0

NetGear 10/100MB **dc1**
 Connected to the INTERNAL LAN

IP Address: 192.168.1.1
Netmask: 255.255.255.0

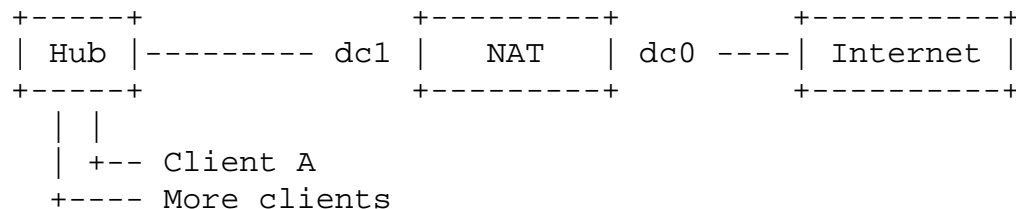
External, Internet-routable IP (provided by ISP, in this example, a cable modem provider)

IP Address: 24.5.0.5
Netmask: 255.255.255.0
Gateway: 24.5.0.1

Local Area Network

In this example, machines on the LAN use the IP addressing scheme 192.168.1.xxx (where xxx is a unique number). There are a variety of different operating systems on the internal LAN including Windows 98, Windows NT, OpenBSD and Linux. Each machine is connected to a hub that is designated for internal use. For this document and its examples the client on the LAN will assume IP address 192.168.1.40

Diagram of Configuration



LEGEND	
NIC dc0	- 24.5.0.5
NIC dc1	- 192.168.1.1
Client A	- 192.168.1.40

6.3.2 Network Address Translation

Introduction to NAT

As more and more businesses and users get on the Internet, each one must have an IP address. Public IP addresses are becoming harder and harder to get. The solution for a lot of people has been Network Address Translation (or "NAT"). NAT is a very simple, yet powerful way to get your LAN connected to the Internet without having to purchase or lease IP addresses for each machine. NAT is also known as "IP Masquerading" if you're a Linux user.

When NAT is up and running correctly, it allows users on the internal LAN to access the Internet through a different IP address (the one you set up with your provider). Each machine on the LAN uses the one IP address (transparently) of the one machine that is set up to use the ISP assigned IP address.

The way NAT works is amazingly simple. When a client on the LAN wants to connect to a machine on the Internet, it sends out a TCP packet with a request to connect. Inside the TCP packet header is the client's IP address (i.e. 192.168.1.40) and the requested host's IP address (i.e. 123.45.67.89). The machine running NAT intercepts this TCP packet and changes the client's IP address from 192.168.1.40 to the IP address of the Internet-connected machine (i.e. 24.5.0.5). This effectively tricks the host machine into thinking the actual connection is from the NAT machine, not the actual client's machine. The host then sends back responses to the NAT machine like it was the one connecting. When the NAT machine receives the responses it quickly translates the destination IP address back from itself to the client's machine and sends the packet to the

client. The client didn't have any idea of what happened and spoofed Internet connectivity is totally transparent.

The example below shows NAT a little more clearly:

```
Client ----- dc1 [ NAT ] dc0 ----- Internet Host
192.168.1.40 --- 192.168.1.1 [ NAT ] 24.5.0.5 --- 123.45.67.89
```

```
OUTGOING TCP Packet                               OUTGOING TCP Packet
From: 192.168.1.40  >>=== NAT ===>>           From: 24.5.0.5
To: 123.45.67.89                                   To: 123.45.67.89
```

```
INCOMING TCP Packet                               INCOMING TCP Packet
From: 123.45.67.89                               From: 123.45.67.89
To: 192.168.1.40  <<=== NAT ===<<           To: 24.5.0.5
```

Why to use NAT

When presented with a cable modem in my new apartment I was also presented with another minor problem. How to get Internet access to my roommates, when the cable modem resides in my room? There were a few options I could implement ranging from obtaining extra IP addresses, to setting up a proxy server, to setting up NAT. (Don't let the cable modem example fool you. NAT is powerful enough to masquerade a large network with hundreds or even thousands of computers!)

There are many reasons why I wanted to set up NAT. The number one reason is for saving money. There are two roommates in my house (each with their own PC) and myself with 3 computers. My ISP only allows for three IP addresses per household. This means that there weren't enough IPs to allow every machine internet access.

By using NAT each machine will have a unique (internal) IP address but share the one IP address given to me by my ISP. The cost goes down.

Setup

In order to enable NAT on your OpenBSD machine you will need to turn on IPF and NAT. This is easily accomplished by editing the files listed below (make the changes to the file so it looks like the options below):

/etc/rc.conf (this file used to start services at boot time)

```
ipfilter=YES
ipnat=YES
```

/etc/sysctl.conf

```
net.inet.ip.forwarding=1
```

After these changes are made, the machine is now ready to for the configuration of NAT.

Configuration

The first step is to configure the IPF rules file (*/etc/ipf.rules*). For the purposes of this document we will allow traffic to pass through this firewall option without any interference. The file should look like this:

```
pass in from any to any
pass out from any to any
```

Again for more information you can read [FAQ 6.2](#)

The NAT configuration file (*/etc/ipnat.rules*) has a very simple syntax. For the configuration set forth above, the file should contain the following entry:

```
map dc0 192.168.1.0/24 -> 24.5.0.5/32 portmap tcp/udp 10000:60000
map dc0 192.168.1.0/24 -> 24.5.0.5/32
```

Here is an explanation for the above lines.

"map"

This is the command you are giving ipnat. It is telling ipnat that this entry is an entry to change IP addresses between the LAN and the Internet.

"dc0"

This is the network interface that is connected to the Internet.

"192.168.1.0/24"

the IP address and netmask (the netmask is in CIDR format). Combined they state "any IP address of value 192.168.1.1 through 192.168.1.254" should be mapped. If you would prefer not to use CIDR notation you can substitute "/24" for "/255.255.255.0".

"24.5.0.5/32"

This IP address and netmask state the IP address that the LAN IP addresses will be mapped to. /32 means one single IP address. You can also map to a /24, or 256 IP addresses (or a /27, or whatever number of bits you'd like)!! This is useful if you have several thousand client machines behind your NAT.... (Of course, this is only useful if that /24 is being routed to your OpenBSD box!)

"portmap tcp/udp 10000:60000"

This maps all tcp/udp packets to ports in the range of 10000 to 60000.

The second line has almost the same entry except for the last portion. This tells ipnat to map anything else (not tcp/udp, those packets are already matched by the first line) to whatever port it requests (used for ICMP, and other protocols). Once this is in the file, all that's needed is to run the IPF daemon.

Running

Executing NAT is a very simple process also. Once the configuration is complete, there are two ways to enable NAT. The first (and best way if possible-to test the setup stage) is to reboot your OpenBSD machine. This is accomplished with the command *"reboot"*

If you would like to run ipnat from the command line, use the following commands:

```
# ipf -Fa -f /etc/ipf.rules -E
# ipnat -CF -f /etc/ipnat.rules
```

The first line is to enable IPF (remember that NAT piggy-backs on IPF therefore IPF must be initialized and running before NAT can be loaded). The options on the command line "-Fa" clear out any existing entries already in effect. "-f /etc/ipf.rules" tells ipf where the rules file can be found. "-E" is the switch to enable the IPF daemon.

The second command line is to enable NAT. "-CF" clears and flushes all existing entries in the NAT table. "-f /etc/ipnat.rules" tells NAT where the NAT rules file is at. NAT is now running. It's as simple as that.

Note: in order to reload the NAT settings (in case you edit the file but don't want to reboot) just execute the 2nd command over again. The settings will be flushed and reloaded.

6.3.3 Nat Knowledge Base

Checking NAT Status

To find out how NAT is doing or make sure the settings have taken effect, you use the "-l" option. This option will list all the settings and current sessions that ipnat is running:

```
# ipnat -l
```

```
map dc0 192.168.1.0/24 -> 24.5.0.5/32 portmap tcp/udp 10000:60000
map dc0 192.168.1.0/24 -> 24.5.0.5/32
```

List of active sessions:

```
MAP 192.168.1.40 2473 <- -> 24.5.0.5 13463 [129.128.5.191 80]
```

The purpose of the first two lines is to confirm the settings that were entered in `/etc/ipnat.rules` earlier. The line(s) below will show you a list of the current NAT controlled connections.

"MAP 192.168.1.40 2473"

This tells you the IP address of the machine on the LAN that is using NAT. The port number used to make the connection is displayed afterwards.

"<- ->"

This shows that NAT is handling the flow of traffic in both directions.

"24.5.0.5 13463"

This denotes that the connection is going to the Internet via IP address 24.5.0.5 and using port 13463.

"129.128.5.191 80"

The IP address and the port being connected to are listed last.

Limitations of NAT (in FTP)

There are a few limitations of NAT. One is with FTP. When a user connects to a remote FTP server and requests information or file, the FTP server will make a connection to the client and transfer the info. This is done on a random free port. This is a problem for users attempting to gain access to FTP servers from within the LAN. When the FTP server sends its information it sends it to the external NIC at a random port. The NAT machine will receive this, but because it has no mappings for the unknown packet and doesn't have any mappings for that port, it will drop the packet and won't deliver it.

The solution to this is to place yourself in "passive mode" in your FTP client. This will tell the server that you want to connect to the server, and not what you just read. Then when you make that connection out NAT will correctly handle your connection.

IP Filter provides another solution for this situation, that is, an ftp proxy which is built-in to the NAT code. To activate it, put something like this before your other NAT mappings.

```
map dc0 192.168.1.0/24 -> 24.5.0.5/32 proxy port ftp ftp/tcp
```

With this in place, the kernel will watch your FTP connections for the "PORT" command coming from the ftp client, and it will replace the IP address and port with it's own outside IP address, and a port of its own choosing. Then it will open up that port and tunnel the data to the port your ftp client asked for. Obviously, this is slightly more resource intensive. But, unless your NAT/IP Filter box is reaching critical mass, you should be fine.

Redirecting Traffic

At times you may need to redirect incoming or outgoing traffic for a certain protocol or port. A good example of this is if there were a server residing inside the LAN running a web server. Incoming connections to your valid Internet IP will find that unless your NAT box is running a web server, no connection can be made. For this purpose we use the NAT 'rdr' directive in the rules file to instruct where to redirect (or route) a particular connection to.

For our example, lets say a web server resides on the LAN with IP address of 192.168.1.80. The NAT rules file needs a new directive to handle this. Add a line similar to the following one to your ipnat.conf:

```
rdr dc0 24.5.0.5/32 port 80 -> 192.168.1.80 port 80
```

The reason for each line is this:

"rdr"

This is the command you are giving ipnat. It is telling ipnat that this entry is an entry to redirect a connection.

"dc0"

This is the network interface that is connected to the Internet.

"24.5.0.5/32"

This means an incoming connection to this IP address (only on dc0, as above)

"port 80"

This is the port (80) that should be redirected. The number "80" didn't have to be used. You can use "port www" also to specify a redirection of port 80. If you would like to use a name instead of a number, the service name and corresponding port, must exist in the file /etc/services.

"192.168.1.80"

The IP address and netmask of the LAN machine which the packets are redirected to. The netmask is always "/32" (and therefore not needed to be specified) so the packets can be redirected to a particular machine.

When the addition is complete reload the NAT rules, and the redirection will start immediately.

NAT versus Proxy

The difference between NAT and an application-based proxy is that the proxy software acts as a middle-man between the Internet and the machines connected on the LAN. This is fine, however each application you want to run on your machine and connect to the Internet through the proxy server **MUST** be proxy-aware (be able to use a proxy server). Not all applications are able to do this (especially games). Furthermore, there simply are not proxy server applications for all of the Internet services out there. NAT transparently maps your internal network so that it may connect to the Internet. The only security advantage to using a proxy software over NAT is that the proxy software may have been made security aware, and can filter based on content, to keep your Windows machine from getting a macro virus, it can protect against buffer overflows to your client software, and more. To maintain these filters is often a high-maintenance job.

6.3.4 Links and X-References

OpenBSD files:

- /etc/ipnat.rules - NAT rules file
- /etc/rc.conf - need to edit to start up ipnat and ipf at boot time
- /etc/sysctl.conf - need to edit to enable IP forwarding
- /usr/share/ipf/nat.1 - samples of ipnat.rules

NAT Internet Links:

- <http://www.openbsd.org/cgi-bin/man.cgi?query=ipnat&sektion=8>
- [Man page showing correct ipnat.rules syntax](#)
- <http://coombs.anu.edu.au/~avalon/>
- <http://www.geektools.com/rfc/rfc1631.txt>

6.4 - DHCP

6.4.1 DHCP Client

To use the DHCP client [dhclient\(8\)](#) included with OpenBSD, edit /etc/hostname.xl0 (this is assuming your main ethernet interface is xl0. Yours might be ep0 or fxp0 or something else!) All you need to put in this hostname file is 'dhcp'

```
# echo dhcp >/etc/hostname.xl0
```

This will cause OpenBSD to automatically start the DHCP client on boot. OpenBSD will gather its IP address, default gateway, and DNS servers from the DHCP server.

If you want to start a dhcp client from the command line, make sure `/etc/dhclient.conf` exists, then try:

```
# dhclient fxp0
```

Where `fxp0` is the interface that you want to receive dhcp on.

No matter how you start the `dhclient`, you can edit the `/etc/dhclient.conf` file to **not** update your DNS according to the dhcp server's idea of DNS by first uncommenting the 'require' lines in it (they are examples of the default settings, but you need to uncomment them to override `dhclient`'s defaults.)

```
request subnet-mask, broadcast-address, time-offset, routers,
       domain-name, domain-name-servers, host-name, lpr-servers, ntp-servers;
```

and then **remove** `domain-name-servers`. Of course, you may want to remove `hostname`, or other settings too.

6.4.2 DHCP Server

If you want to use OpenBSD as a DHCP server [dhcpd\(8\)](#), edit `/etc/rc.conf`. Set it up so that `dhcpd_flags="-q"` instead of `dhcpd_flags=NO`. Put the interfaces that you want `dhcpd` to **listen** on in `/etc/dhcpd.interfaces`.

```
# echo x11 x12 x13 >/etc/dhcpd.interfaces
```

Then, edit `/etc/dhcpd.conf`. The options are pretty self explanatory.

```
option domain-name "xyz.mil";
option domain-name-servers 192.168.1.3, 192.168.1.5;

subnet 192.168.1.0 netmask 255.255.255.0 {
    option routers 192.168.1.1;

    range 192.168.1.32 192.168.1.127;
}
```

This will tell your dhcp clients that the domain to append to DNS requests is `xyz.mil` (so, if the user types in 'telnet joe' then it will send them to `joe.xyz.mil`). It will point them to DNS servers `192.168.1.3` and `192.168.1.5`. For hosts that are on the same network as an ethernet interface on the OpenBSD machine, which is in the `192.168.1.0/24` range, it will assign them an IP address between `192.168.1.32` and `192.168.1.127`. It will set their default gateway as `192.168.1.1`.

If you want to start `dhcpd` from the command line, after editing `/etc/dhcpd.conf`, try:

```
# dhcpd -q fxp0
```

Where `fxp0` is an interface that you want to start serving dhcp on. The `-q` flag makes `dhcpd` quiet, otherwise it is very noisy.

If you are serving DHCP to a Windows box, you may want to `dhcpd` to give the client a 'WINS' server address. To make this happen, just the following line to your `/etc/dhcpd.conf`:

```
option netbios-name-servers 192.168.92.55;
```

(where `192.168.92.55` is the IP of your Windows or Samba server.) See [dhcp-options\(5\)](#) for more options that your DHCP clients may want.

6.5 - PPP

Point-to-Protocol is generally what is used to create a connection to your ISP via your modem. OpenBSD has 2 ways of doing this.

- [pppd\(8\)](#) - Which is the kernel ppp daemon.
- [ppp\(8\)](#) - Which is the userland ppp daemon.

The first one we will cover will be the userland PPP daemon. To start off you will need some simple information about your

isp. Here is a list of helpful information that you will need.

- Your ISP's dialup number
- Your nameserver
- Your username and password.
- Your gateway

Some of these you can do without, but would be helpful in setting up your ppp. The userland PPP daemon uses the file [/etc/ppp/ppp.conf](#) as its configuration file. There are many helpful files in **/etc/ppp** that can have different setups for many different situations. You should take a browse though that directory.

Also, make sure, that if your not using a GENERIC kernel, that you have this line in your configuration file:

```
pseudo-device tun 2
```

Initial Setup - for PPP(8)

Initial Setup for the userland PPP daemon consists of editing your **/etc/ppp/ppp.conf** file. This file doesn't exist by default, but there is a file **/etc/ppp/ppp.conf.sample** in which you can simply edit to create your own **ppp.conf** file. Here I will start with the simplest setup and probably most used setup. Here is a quick **ppp.conf** file that will simply connect to your ISP and set your default routes and nameserver. With this file all the information you need is your ISP's phone number and your username and password.

```
default:
set log Phase Chat LCP IPCP CCP tun command
set device /dev/cua01
set speed 115200
set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \"\" AT OK-AT-OK ATE1Q0
OK\\dATDT\\t TIMEOUT 40 CONNECT"
```

NOTICE - On OpenBSD 2.6, the system shipped with a */etc/ppp/ppp.conf.example* that had an incorrect setting for device. The device was "set device /dev/cuaa0". This should be /dev/cua00 which would correspond to serial device 1. (COM1). Your device might not be on COM1, however the nameing scheme was wrong.

The section under the **default:** tag will get executed each time. Here we setup all our critical information. Here with "set log" we set our logging levels. This can be changed, refer to [ppp\(8\)](#) for more info on setting up logging levels. Our device gets set with "set device". This is the device that the modem is on. In this example the modem is on com port 2. Therefore com port 1 would be /dev/cua00. With "set speed" we set the speed of our dialup connection and with "set dial" we set our dialup parameters. With this we can change our timeout time, etc. This line should stay pretty much as it is though.

Now we can move on and setup our information specific to our ISP. We do this by adding another tag under our **default:** section. This tag can be called anything you want, easiest to just use the name of your ISP. Here I will use **myisp:** as our tag referring to our ISP. Here is a simple setup incorporating all we need to get ourselves connected.

```
myisp:
set phone 1234567
set login "ABORT NO\\sCARRIER TIMEOUT 5 ogin:--ogin: ppp word: ppp"
set timeout 120
set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
add default HISADDR
enable dns
```

Here we have setup essential info for that specific ISP. The first option "set phone" sets your ISP's dialup number. The "set login" sets our login options. Here we have the timeout set to 5, this means that we will abort our login attempt after 5 seconds if no carrier. Otherwise it will wait for "login:" to be sent and send in your username and password. In this example our Username = ppp and Password = ppp. These values will need to be changed. The line "set timeout" sets the timeout for the entire login process to 120 seconds. The "set ifaddr" line is a little tricky. Here is a more extensive explanation.

```
set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.0 0.0.0.0
```

In the above line, we have it set in the format of "**set ifaddr [myaddr[/nn] [hisaddr[/nn] [netmask [triggeraddr]]]**". So the first IP specified is what we want as our IP. If you have a static IP address, you set it here. In our example we use /0 which says that no bits of this ip address need to match and the whole thing can be replaced. The second IP specified is what we expect as their IP. If you know this you can specify it. Again in our line we don't know what will be assigned, so we let them tell us. The third option is our netmask, here set to 255.255.255.0. If triggeraddr is specified, it is used in place of myaddr in the initial IPCP negotiation. However, only an address in the myaddr range will be accepted. This is useful when negotiating with some PPP implementations that will not assign an IP number unless their peer requests ``0.0.0.0".

The next option used "add default HISADDR" sets our default route to their IP. This is 'sticky', meaning that if their IP should change, our route will automatically be updated. With "enable dns" we are telling our ISP to authenticate our nameserver addresses. Do NOT do this if you are running an local DNS, as ppp will simply circumvent its use by entering some nameserver lines in /etc/resolv.conf.

Using PPP(8)

Now that we have our **ppp.conf** file setup we can start trying to make a connection to our ISP. I will detail some commonly used arguments with ppp.

- **ppp -auto myisp** - This will run ppp, configure your interfaces and connect to your isp and then go into the background.
- **ppp -ddial myisp** - This is similar to -auto, but if your connection is dropped it will try and reconnect.

By using **/usr/sbin/ppp** with no options will put you into interactive mode. From here you can interact directly with the modem, it is great for debugging problems in your **ppp.conf** file.

ppp(8) extra's

In some situations you might want commands executed as your connection is made or dropped. There are two files you can create for just these situations. **/etc/ppp/ppp.linkup** and **/etc/ppp/ppp.linkdown**. Sample configurations can be viewed here:

- [ppp.linkup](#)
- [ppp.linkdown](#)

Extended information can be found at <http://www.freebsd.org/handbook/userppp.html> or <http://www.freebsd.org/faq/userppp.html>.

6.6 - Tuning networking parameters

6.6.1 - How can I tweak the kernel so that there are a higher number of retries and longer timeouts for TCP sessions?

You would normally use this to allow for routing or connection problems. Of course, for it to be most effective, both sides of the connection need to use similar values.

To tweak this, use `sysctl` and increase the values of:

```
net.inet.tcp.keepintime
net.inet.tcp.keeptime
net.inet.tcp.keepidle
net.inet.tcp.keepintvl
```

Using `sysctl -a`, you can see the current values of these (and many other) parameters. To change one, use `sysctl -w`, as in `sysctl -w net.inet.tcp.keeptime=28800`.

6.6.2 - How can I turn on directed broadcasts?

Normally, you don't want to do this. This allows someone to send traffic to the broadcast address(es) of your connected network(s) if you are using your OpenBSD box as a router.

There are some instances, in closed networks, where this may be useful, particularly when using older implementations of the NetBIOS protocol. This is another `sysctl`. `sysctl -w net.inet.ip.directed-broadcast=1` turns this on. Read about [smurf attacks](#) if you want to know why it is off by default.

6.6.3 - I don't want to the kernel to dynamically allocate a certain port

There is a `sysctl` for this also. From [sysctl\(8\)](#):

Set the list of reserved TCP ports that should not be allocated by the kernel dynamically. This can be used to keep daemons from stealing a specific port that another program needs to function. List elements may be separated by commas and/or whitespace.

```
sysctl -w net.inet.tcp.baddynamic=749,750,751,760,761,871
```

It is also possible to add or remove ports from the current list.

```
sysctl -w net.inet.tcp.baddynamic+=748
sysctl -w net.inet.tcp.baddynamic=-871
```

6.7 - Simple NFS usage

NFS, or Network File System, is used to share a filesystem over the network. A few choice man pages to read before trying to setup a NFS server are:

- [nfsd\(8\)](#)
- [mountd\(8\)](#)
- [exports\(5\)](#)

This section will go through the steps for a simple setup of NFS. This example details a server on a LAN, with clients accessing NFS on the LAN. It does not talk about securing NFS. We presume you have already setup packet filtering or other firewalling protection, to prevent outside access. If you are allowing outside access to your NFS server, and you have any kind of sensitive data stored on it, we strongly recommend that you employ [IPSec](#). Otherwise, people can potentially see your NFS traffic. Someone could also pretend to be the IP address which you are allowing into your NFS server. There are several attacks that can result. When properly configured, IPSec protects against these types of attacks.

Another important security note. Don't just add a filesystem to `/etc/exports` without some kind of list of allowed host(s). Without a list of hosts which can mount a particular directory, anyone on who can reach your host will be able to mount your NFS exports.

The setup consists of a server with the ip **10.0.0.1**. This server will be serving NFS only to clients within that network. The first step to setting up NFS is to setup your `/etc/exports` file. This file lists which filesystems you wish to have accessible via NFS and defines who is able to access them. There are many options that you can use in your `/etc/exports` file, and it is best that you read the [exports\(5\)](#) man page. For this example we have an `/etc/exports` that looks like this:

```
#
# NFS exports Database
# See exports(5) for more information. Be very careful, misconfiguration
# of this file can result in your filesystems being readable by the world.
/work -alldirs -ro -network 10.0.0 -mask 255.255.255.0
```

This means that the local filesystem **/work** will be made available via NFS. **-alldirs** specifies that clients will be able to mount at any point under the **/work** mount point. **-ro** specifies that it will only be allowed to be mounted read-only. The last two arguments specify that only clients within the 10.0.0.0 network using a netmask of 255.255.255.0 will be authorized to mount this filesystem. This is important for some servers that are accessible by different networks.

Once your */etc/exports* file is setup, you can go ahead and setup your NFS server. You should first make sure that options **NFSSERVER** & **NFSCIENT** are in your kernel configuration. (GENERIC kernel has these options included.) Next, you should set **nfs_server=YES** in */etc/rc.conf*. This will bring up both **nfsd(8)** and **mountd(8)** when you reboot. Now, you can go ahead and start the daemons yourself. These daemons need to be started as root, and you need to make sure that **portmap(8)** is running on your system. Here is an example of starting **nfsd(8)** which serves on both TCP and UDP using 4 daemons. You should set an appropriate number of NFS server daemons to handle the maximum number of concurrent client requests that you want to service.

```
# /sbin/nfsd -tun 4
```

Not only do you have to start the **nfsd(8)** server, but you need to start **mountd(8)**. This is the daemon that actually services the mount requests on NFS. To start **mountd(8)**, simply type:

```
# /sbin/mountd
```

If you make changes to */etc/exports* while NFS is already running, you need to make **mountd** aware of this! Just HUP it:

```
# kill -HUP `cat /var/run/mountd.pid`
```

Checking Stats on NFS

From here, you can check to make sure that all these daemons are up and registered with RPC. To do this, use **rpcinfo(8)**.

```
$ rpcinfo -p 10.0.0.1
  program vers proto  port
  100000    2    tcp    111  portmapper
  100000    2    udp    111  portmapper
  100005    1    udp    633  mountd
  100005    3    udp    633  mountd
  100005    1    tcp    916  mountd
  100005    3    tcp    916  mountd
  100003    2    udp    2049 nfs
  100003    3    udp    2049 nfs
  100003    2    tcp    2049 nfs
  100003    3    tcp    2049 nfs
```

During normal usage, there are a few other utilities that allow you to see what is happening with NFS. One is [showmount\(8\)](#), which allows you to view what is currently mounted and who is mounting it. There is also **nfsstat(8)** which shows much more verbose statistics. To use **showmount(8)**, try **/usr/bin/showmount -a host**. For example:

```
$ /usr/bin/showmount -a 10.0.0.1
All mount points on 10.0.0.1:
10.0.0.37:/work
```

Mounting NFS Filesystems

NFS filesystems should be mounted via **mount(8)**, or more specifically, [mount_nfs\(8\)](#). To mount a filesystem **/work** on host 10.0.0.1 to local filesystem **/mnt**, do this (note that you don't need to use an IP address, **mount** will resolve host names):

```
# mount -t nfs 10.0.0.1:/work /mnt
```

To have your system mount upon boot, add something like this to your */etc/fstab*:

```
10.0.0.1:/work /mnt nfs rw 0 0
```

It is important that you use **0 0** at the end of this line so that your computer does not try to **fsck** the NFS filesystem on

boot!!!! The other standard security options, such as noexec, nodev, and nosuid, should also be used where applicable. Such as:

```
10.0.0.1:/work /mnt nfs rw,nodev,nosuid 0 0
```

This way, no devices or setuid programs on the NFS server can subvert security measures on the NFS client. If you are not mounting programs which you expect to run on the NFS client, add noexec to this list.

6.8 - Domain Name Service - DNS, BIND, and named

6.8.1 What is DNS?

Domain Name Service is a network facility allowing IP network domains to provide name-to-IP address resolution and IP address-to-name resolution in response to a query. Your OpenBSD installation is configured by default as a DNS client but not as a DNS server. That is, your OpenBSD installation can perform a DNS query against a domain name server for the address of a machine, but it cannot answer such DNS queries itself unless you specifically configure it to do so.

My OpenBSD machine is currently connected to the Internet via my ISP, so I can use the [nslookup\(8\)](#) utility to execute the DNS query:

```
$ nslookup www.openbsd.org
Server: ns4.us.prserv.net
Address: 165.87.201.244
```

```
Non-authoritative answer:
Name: www.openbsd.org
Address: 129.128.5.191
```

165.87.201.244 is the name server which answered, because it is the nameserver that my ISP told me to use with my account and whose number is entered in [/etc/resolv.conf](#). But the answer was not authoritative. For an authoritative answer, let's find which is the authoritative DNS server for the *openbsd.org* domain and ask it for the address of *www.openbsd.org*:

```
# Identify the name servers for openbsd.org
# with the help of my ISP's name server.
$ nslookup -type=NS openbsd.org
Server: ns4.us.prserv.net
Address: 165.87.201.244
```

```
Non-authoritative answer:
openbsd.org nameserver = cvs.openbsd.org
openbsd.org nameserver = gandalf.sigmasoft.com
openbsd.org nameserver = cs.colorado.edu
openbsd.org nameserver = ns.appli.se
openbsd.org nameserver = zeus.theos.com
```

```
Authoritative answers can be found from:
cvs.openbsd.org internet address = 199.185.137.3
gandalf.sigmasoft.com internet address = 198.144.202.98
cs.colorado.edu internet address = 128.138.243.151
ns.appli.se internet address = 194.198.196.230
zeus.theos.com internet address = 199.185.137.1
```

```
# Use the info gained to query for an authoritative
# resolution: query the authoritative zeus.theos.com.
$ nslookup www.openbsd.org zeus.theos.com
Server: zeus.theos.com
```

Address: 199.185.137.1

Name: www.openbsd.org

Address: 129.128.5.191

zues.theos.com is, one would suppose, running OpenBSD and is properly configured to be a DNS server for the *openbsd.org* domain.

6.8.1.1 Where can I learn all about DNS and its implementation under OpenBSD?

- See RFC's [1033](#), [1034](#), and [1035](#) for more information on the Internet name-domain system.
- Read the O'Reilly Associates book *DNS and BIND* .
- Read the [OpenBSD Manual](#) especially the pages for
 - [dig\(1\)](#)
 - [nslookup\(8\)](#)
 - [gethostbyname\(3\)](#)
 - [named\(8\)](#)
 - [resolver\(3\)](#)
 - [resolver\(5\)](#)

The [dig\(1\)](#) command is especially useful, because it can query a domain and return information in much the same record format as required in BIND configuration files. You can use [dig\(1\)](#) to examine name servers you know to be operating properly as a way of comparing your setup to theirs.

6.8.2 Does my machine need to be a domain name server?

If you aren't sure that you need your machine to perform the role of DNS server, don't configure it as one. The OpenBSD installation does not, by default, activate your machine as a domain name server, though all necessary files are installed. For most workstations, just the [/etc/hosts](#) file naming local machines' IP addresses and the [/etc/resolv.conf](#) file for indicating which DNS servers serve you out on the intranet or internet is sufficient.

On the other hand, you might need to set up a machine as a domain name server:

- If you have an IP LAN on which you do not wish to replicate "hosts" files of local addresses machine by machine. In such a case, you may configure your OpenBSD machine as a DNS server and serve queries from the other machines on your LAN.
 - **Note:** There is no practical restriction on the number of DNS servers on a LAN. Any or all machines on the LAN may offer DNS service if they are so configured. Whether any such server is considered authoritative from outside your LAN (or is even known from outside your LAN) is a configuration factor which typically is controlled at the next level up from your LAN in the domain hierarchy.
- If you have an IP LAN on which reside machines you will wish to be findable via DNS query by machines on another IP LAN or WAN.
- If you experience difficulties resolving the local hostname to an IP address, or resolving other local names to IP addresses even though you have correct */etc/hosts* and */etc/resolv.conf* (E.g., Netscape on OpenBSD sometimes exhibits this behavior because it implements its own resolver instead of just using *gethostbyname(3)* to look up addresses.)

One more consideration is speed of execution. Since name resolution is an iterative process, in which the name server makes repeated queries to other nameservers for addresses in remote domains, name resolution may take slightly longer if you have a modem connection to the Internet and are querying your own DNS server for remote addresses (which will then iteratively query remote name servers via the modem) than if you are querying your ISP's name server (which probably has a faster connection to remote name servers).

6.8.3 What are the software components of the DNS server?

- `named` ("*name daemon*")
- Configuration files in the directory hierarchy under `/var/named/`

6.8.3.1 What level of BIND is supported?

BIND is the name of the behavioral specification of a domain name server. Domain name server components exist to collectively implement BIND.

There are two distinct BIND specifications:

1. BIND 4
2. BIND 8

As installed, OpenBSD **named** supports BIND 4.x.

6.8.3.2 What are some of the alternatives to providing DNS via the default BIND 4.x implementation?

- The BIND 8.x implementation in `/usr/ports/net/bind8`. (See [ports](#))
- D. J. Bernstein's [DNSCache](#). DNSCache is also available in the ports tree at `/usr/ports/net/dnscache/`. This program includes a minimalist DNS server which is perfect for sites who do not have a large existing database of DNS entries. DNSCache could easily be script driven. Its setup does not resemble anything below, but is well documented at its web page.
- Some people who were particularly unhappy with BIND decided to write their own DNS server from scratch, [Dents](#). It contains some interesting features.

6.8.3.2.1 Security note

If you use these alternative implementations of domain name service, you are providing a critical network service using software which may not have been subject to quite the same level of scrutiny as the [security-audited named](#) name daemon in the base installation. This is a significant consideration, since if a domain name server is compromised, resolvers using that name server can be re-directed to impostor sites.

6.8.4 How much do I have to install?

If the default networking setup was installed properly at OpenBSD installation time, everything is already installed. You just have to configure the name daemon ("`named`").

6.8.5 How do I configure DNS?

You configure OpenBSD DNS by editing and/or creating files that control the name daemon `named`. These files reside by default in the directory `/var/named` and its subdirectories, especially the file `/var/named/named.boot` which is the initialization file for **named**. There are also a couple of other configuration steps to be taken in `/etc`.

In this document, we will configure the name daemon on `nemo.yewtopia.com` be the primary nameserver for the (very small!) domain `yewtopia.com`. The address of `nemo.yewtopia.com` is `192.168.1.9`. Two other machines are on that subnet, `crater.yewtopia.com` at `192.168.1.1` and `earhart.yewtopia.com` at `192.168.1.2`.

6.8.5.1 Configuration in `/etc/named`

6.8.5.1.1 `/var/named/named.boot`

```
; tell what subdir has the lookup database files
directory          /namedb
```

```

; type      domain                source host/file
backup file
cache
root.cache
primary    0.0.127.IN-ADDR.ARPA  localhost.rev

; example primary server config:
primary    yewtopia.com yewtopia
primary    1.168.192.IN-ADDR.ARPA yewtopia.rev

```

This tells the initialization process in what subdirectory and under which filenames to find the configuration files for *yewtopia.com*.

6.8.5.1.2 /var/named/namedb/localhost.rev

```

; Reverse lookup for localhost interface
@          IN          SOA          nemo.yewtopia.com.
your_id.nemo.yewtopia.com. (
                                14          ; Serial
                                3600         ; Refresh
                                900          ; Retry
                                3600000     ; Expire
                                3600 )      ; Minimum
1          IN          NS           nemo.yewtopia.com.
1          IN          PTR          localhost.yewtopia.com.

```

6.8.5.1.3 /var/named/namedb/yewtopia

```

; yewtopia.com domain database
yewtopia.com.          IN          SOA          nemo.yewtopia.com.
your_id.nemo.yewtopia.com. (
                                14          ; Serial
                                3600         ; Refresh
                                900          ; Retry
                                3600000     ; Expire
                                3600 )      ; Minimum
                                IN          NS           nemo.yewtopia.com.

; Addresses
localhost.yewtopia.com.      IN A      127.0.0.1
crater.yewtopia.com.        IN A      192.168.1.1
earhart.yewtopia.com.       IN A      192.168.1.2
nemo.yewtopia.com.          IN A      192.168.1.9

```

6.8.5.1.4 /var/named/namedb/yewtopia.rev

```

; yewtopia domain reverse lookup database
1.168.192.in-addr.arpa. IN          SOA          nemo.yewtopia.com.
your_id.nemo.yewtopia.com. (
                                14          ; Serial
                                3600         ; Refresh
                                900          ; Retry
                                3600000     ; Expire
                                3600 )      ; Minimum

```

```
1.168.192.in-addr.arpa. IN      NS      nemo.yewtopia.com.
```

```
; Addresses
```

```
1.1.168.192.in-addr.arpa. IN PTR crater.yewtopia.com.
```

```
2.1.168.192.in-addr.arpa. IN PTR earhart.yewtopia.com.
```

```
9.1.168.192.in-addr.arpa. IN PTR nemo.yewtopia.com.
```

6.8.5.2 Configuration in */etc*

6.8.5.2.1 */etc/resolv.conf*

Make sure */etc/resolv.conf* now points to the domain of local machine (instead of, for example, your ISP's name server) so that name resolution requests actually get sent to the **named** you have configured!

```
domain yewtopia.com
lookup file bind
```

6.8.5.2.2 */etc/hosts*

If you previously had added the addresses of various machines to the */etc/hosts* file, you might consider shortening your */etc/hosts* file back to the default:

```
# Host addresses
127.0.0.1      localhost      localhost.localdomain
192.168.1.9    nemo           nemo.yewtopia.com
```

So that **named** isn't bypassed in favor of (possibly outdated) addresses in the */etc/hosts* file. Make sure you have at least the default *localhost* entry or your network won't start properly! Note also *nemo* must appear in its own hosts file or you will see a (mostly harmless) error message at bootup when */etc/netstart* invokes [route\(8\)](#) in order to add *nemo* (whose name appears in */etc/myname*).

6.8.5.3 Using [dig\(1\)](#) to examine the results.

```
$ dig @nemo.yewtopia yewtopia any any

; <<>> DiG 2.2 <<>> @nemo.yewtopia yewtopia any any
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59904
;; flags: qr rd ra; Ques: 1, Ans: 2, Auth: 0, Addit: 1
;; QUESTIONS:
;;      yewtopia, type = ANY, class = ANY

;; ANSWERS:
yewtopia.      3600    SOA      nemo.yewtopia.
your_id.nemo.yewtopia. (
                14      ; serial
                3600   ; refresh (1 hour)
                900    ; retry (15 mins)
                3600000 ; expire (41 days 16 hours)
                3600   ) ; minimum (1 hour)
yewtopia.      3600    NS       nemo.yewtopia.

;; ADDITIONAL RECORDS:
nemo.yewtopia. 3600    A        192.168.1.9
```

```
;; Total query time: 4 msec
;; FROM: nemo to SERVER: nemo.yewtopia. 192.168.1.9
;; WHEN: Tue May  2 23:47:19 2000
;; MSG SIZE  sent: 25  rcvd: 102
```

6.8.6 How and when do I start and stop DNS?

6.8.6.1 Starting DNS

The name daemon **named** is launched during system startup from [/etc/rc](#) if the line installed by default in [/etc/rc.conf](#).

```
named_flags=NO          # for normal use: ""
```

is changed to

```
named_flags=""         # for normal use: ""
```

Also, examine these lines in [/etc/rc.conf](#):

```
named_user=named       # Named should not run as root unless necessary
named_chroot=/var/named # Where to chroot named if not empty
```

These defaults will be correct for nearly all setups.

To start **named** by hand, use the [ndc\(8\)](#) command. For example:

```
# ndc start
      or
# ndc restart
```

6.8.6.2 Stopping DNS

The best way to stop the name daemon is to use the [ndc\(8\)](#) command. For example:

```
# ndc stop
```

If this fails to work, find the process id of **named** and use the [kill\(1\)](#) command to end that process. The PID for **named** while it is running is found as the first line in the file [/var/named/named.pid](#)

```
# cat /var/named/named.pid
4608
named -t /var/named -u named
# kill -KILL 4608
```

6.8.6.3 Restarting DNS with an altered configuration

To cause a running instance of the name daemon to restart itself reloading its configuration after you have made changes, send it a "hangup" signal:

```
# kill -HUP 4608
```

or by using the [ndc\(8\)](#) command. For example:

```
# ndc restart
```

6.8.7 What didn't you tell me about setting up DNS?

There's a lot we didn't tell you, for example, how to set up DNS so that queries for intranet domains that aren't visible from the root of the domain hierarchy get relayed to servers within your enterprise. Read the [documents we recommended](#) for more information on DNS.



www@openbsd.org

\$OpenBSD: faq6.html,v 1.73 2000/05/26 19:43:44 reinhard Exp \$

7.0 - Keyboard Controls

Table of Contents

- [7.1 - How do I remap the keyboard?](#)
 - [7.2 - Is there gpm or the like in OpenBSD?](#)
 - [7.3 - How do I clear the console each time a user logs out?](#)
 - [7.4 - Accessing the console scrollback buffer.](#)
-

7.1 - How do I remap the keyboard? (*pcvt/i386 specific*)

By using **kcon(1)**, "kcon - keyboard control and remapping for the pcvt driver" Before you can use this, you need to turn on character mapping with scon.

Example:

```
bsd# scon -o
bsd# kcon -m gb
```

This will load the keycap file for Great Britain. Note that if you run scon -o again, this will turn character mapping back off, which you probably don't want to do if you are using kcon for remapping. Keep this in mind if you use scon in any automated fashion, such as in scripts.

These commands can only be executed from a pcvt virtual terminal.

7.2 - Is there gpm or the like in OpenBSD?

Not yet. If you are willing to do a port, contact the [ports mailing list](#).

7.3 - Clearing the console each time a user logs out.

To do this you must add a line in /etc/gettytab. Change the section currently:

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

Add the line ":cl=\E[H\E[2J:" at the end, so it ends up looking like this:

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

7.4 - Accessing the Console Scrollback Buffer (*pcvt/i386 specific*)

For OpenBSD 2.6 the ability to scrollback on the console was added for i386 machines. This allows you to see information that has already scrolled past your screen. To be able to move up and down in the buffer simply use the key combination "[shift]+[pgup]" to move up, and "[shift]+[pgdn]". The default scrollback buffer (the amount of pages that you can move up and view) is 8, but this can be changed via the [scon\(1\)](#) utility. For example, to change the number of pages you can scrollback to 10, do this:

```
bsd$ scon -b 10
```

This command can only be executed from a pcvt virtual terminal.

[\[Back to Main Index\]](#) [\[To Section 6.0 - Networking\]](#) [\[To Section 8.0 - General Questions\]](#)



www@openbsd.org

\$OpenBSD: faq7.html,v 1.13 2000/01/22 17:40:31 ericj Exp \$

8.0 - General Questions

Table of Contents

- [8.1 - What are these kerberos warnings when I first login?](#)
 - [8.2 - How do I change virtual terminals? \(I386 ONLY\)](#)
 - [8.3 - I forgot my root password..... What do I do!](#)
 - [8.4 - X won't start, I get lots of error messages](#)
 - [8.5 - What is CVS, and how do I use it?](#)
 - [8.6 - What is the ports tree?](#)
 - [8.7 - What are packages?](#)
 - [8.8 - Is there any way to use my floppy drive if it's not attached during boot?](#)
 - [8.9 - OpenBSD Bootloader \(*i386 specific*\)](#)
 - [8.10 - Using S/Key on your OpenBSD system](#)
 - [8.11 - Why is my Macintosh losing so much time?](#)
 - [8.12 - Will OpenBSD run on multiprocessor machines?](#)
 - [8.13 - I sometimes get Input/output error when trying to use my tty devices](#)
 - [8.14 - Where can I find a Netscape binary for OpenBSD?](#)
 - [8.15 - How do I use the mg editor?](#)
 - [8.16 - Ksh does not appear to read my .profile!](#)
 - [8.17 - Why does my /etc/motd file get written over when I modified it?](#)
 - [8.18 - Why does www.openbsd.org run on Solaris?](#)
-

8.1 - What are these Kerberos warning when I first login?

When you first install your system you will most likely notice a warning message that is something like this:

```
Warning: no Kerberos tickets issued.
```

THIS WARNING IS COMPLETELY IRRELEVANT AND SHOULD ONLY BE REMOVED IF COMPLETELY NECESSARY

Well since you probably haven't set up Kerberos on your system, you wouldn't be getting a ticket. If you do have Kerberos running, you need to check into that. See [faq10 Kerberos Setup](#) FAQ. If you can't STAND that warning and never plan on using Kerberos here is how to get rid of it for good.

- Make sure you have the source code.
- Edit `/usr/share/mk/bsd.own.mk` and set `KERBEROS` to 'no', or create an `/etc/mk.conf` file that does this.
- `cd /usr/src/usr.bin/login ; make clean ; make ; make install`

8.2 - How do I change virtual terminals?

Simply type [ctrl] - [alt] - [One of the function keys] or just hit [f9]-[f12] which are just mapped to [ctrl] - [alt] functions. (i386)

Only the i386 arch has virtual terminal capabilities. You can also use virtual terminals when using X. For example, If you start X on term 1, and switch with [ctrl]-[alt]-[F2] to term 2, X will seem to disappear and will not come back by simply switching back to term 1. You must [ctrl]-[alt]-[F5] to get your X display back.

8.3 - I forgot my root password, what do I do now?

A few steps to recovery

1. Boot into single user mode. For i386 arch type boot -s at the boot prompt.
2. mount the drives.

```
bsd# fsck -p / && mount -u /
```
3. If /usr is not the same partition that / is (and it shouldn't be) then you will need to mount it, also

```
bsd# fsck -p /usr && mount /usr
```
4. run [passwd\(1\)](#)
5. boot into multuser mode.. and *remember* your password!

8.4 - X won't start, I get lots of error messages

If you have X completely set up and you are using an XF86Config that you know works then the problem most likely lies in the machdep.allowaperture. You also need to make sure that both:

```
option XSERVER
option APERTURE
```

are in your kernel configuration. [BOTH of these are in the GENERIC kernel]

Then you need to edit `/etc/sysctl.conf` and set **machdep.allowaperture=1**. This will allow X to access the aperture driver. This would be set up if the question during install about whether or not you would be running X was answered correctly. OpenBSD requires for all X servers that the aperture driver be set, because it controls access to the I/O ports on video boards.

For other X problems on the i386, consult the XFree86 FAQ at <http://www.xfree86.org/FAQ/>.

8.5 - What is CVS? and How do I use it?

CVS is what the OpenBSD project uses to control changes to the source code. CVS stands for Concurrent Versions System. You can read more about CVS at <http://www.cyclic.com/>. CVS can be used by the end user to keep up to date with source changes, and changes in the ports tree. CVS makes it extremely simple to download the source via one of the many CVS mirrors for the project.

How to initially setup your CVS environment

There are a few ways to initially set up your CVS environment. To start off, you will need an initial CVS checkout of the sources. If you bought the CD, you're in luck, because it holds the CVS checkout for that release. You can extract it from your CD by doing one of the following:

- 1) copy the tree off it, (assuming the CD is mounted on /mnt):

```
# mkdir /usr/src
# cd /mnt; cp -Rp CVS Makefile bin distrib etc games gnu \
include kerberosIV lib libexec lkm regress sbin share \
sys usr.bin usr.sbin /usr/src
```

- 2) Or, alternatively, use a union mount with the CD below a writable directory. However, be aware that the union filesystem code is not flawless.

```
# mkdir /usr/src
# mount -t union -o -b /mnt /usr/src
```

after this, /usr/src will be a nice checkout area where all cvs(1) commands will work OK.

If you don't have an OpenBSD CD, you will have to retrieve the sources from one of the OpenBSD AnonCVS servers. These servers are listed on <http://www.openbsd.com/anoncv.html> Once you have chosen a server you need to choose which module you are going to retrieve. There are three main modules available for checkout from the CVS tree. These are:

- *src* - The src module has the complete source code for OpenBSD. This includes userland and kernel sources.
- *ports* - The ports module holds all you need to have the complete OpenBSD ports tree. To read more on the OpenBSD ports tree, read [Section 8.6](#) of the OpenBSD FAQ.
- *www* - The www module contains all OpenBSD web pages, including this FAQ.

Now that you have decided which module that you wish to retrieve, there is one more step left before you can retrieve it. You must decide which method to use. CVS by default retrieves files using rsh(1), but some AnonCVS servers don't allow for this so in most cases it's best to use ssh. For those of you behind a firewall there are also the options of pserver and some AnonCVS servers run ssh on port 222. Be sure to check <http://www.openbsd.com/anoncv.html> for which servers support what protocols. Next I will show how to do a simple source checkout. Here I will be using an AnonCVS server located in the U.S., but remember that if you are outside of the U.S you need to use a server that is located nearby. There are many AnonCVS servers located throughout the world, so choose one nearest you. I will also be using ssh to retrieve the files.

```
ericj@oshibana:~> export CVS_RSH=/usr/bin/ssh
ericj@oshibana:~> echo $CVS_RSH
/usr/bin/ssh
ericj@oshibana:~> export CVSROOT=anoncv@anoncv.usa.openbsd.org:/cvs
ericj@oshibana:~> cvs get src
Warning: Remote host denied X11 forwarding, perhaps xauth program could not be run on
the server side.
cvs checkout: in directory src:
cvs checkout: cannot open CVS/Entries for reading: No such file or directory
cvs server: Updating src
U src/Makefile
[snip]
```

Notice here also that I set the CVSROOT environmental variable. This is the variable that tells cvs(1) which AnonCVS server to use. This can also be specified using the *-d* option. For example:

```
ericj@oshibana:~>cvs -d anoncv@anoncv.usa.openbsd.org:/cvs get src
```

These commands should be run in /usr, which will then create the directories of /usr/src, /usr/ports, and /usr/www. Depending, of course, on which module you checkout. You can download these modules to anywhere, but if you wanted to do work with them (ie make build), it is expected that they be at the place above.

Keeping your CVS tree up-to-date

Once you have your initial tree setup, keeping it up-to-date is the easy part. You can update your tree at any time you choose, some AnonCVS servers update more often than others, so again check <http://www.openbsd.com/anoncv.html>. In this example I will be updating my www module from anoncv.usa.openbsd.org. Notice the *-q* option that I use, this makes the output not so verbose coming from the server.

```
ericj@oshibana:~> echo $CVSROOT
anoncv@anoncv.usa.openbsd.org:/cvs
ericj@oshibana:~> cvs -q up -PAd www
Warning: Remote host denied X11 forwarding, perhaps xauth program could not be run on
the server side.
U www/want.html
M www/faq/faq8.html
ericj@oshibana:~>
```

Other cvs options

For some, bandwidth and time are serious problems when updating repositories such as these. So CVS has a `-z[1-9]` option which uses gzip to compress the data. To use it, do `-z[compression-level]`, for instance, `-z3` for a compression level of 3.

8.6 - What is the ports tree?

The ports tree is a set of Makefiles that download, patch, configure and install userland programs so you can run them in OpenBSD environment without having to do all that by hand. You can get the ports tree from any of the OpenBSD ftp servers in `/pub/OpenBSD/2.6/ports.tar.gz`. The most recent ports are available via the 'ports' cvs tree, or `/pub/OpenBSD/snapshots/ports.tar.gz`. For most of you however, packages will be a much better option. Packages are created from ports and are already compiled and ready to use. To read more on packages read [FAQ8.7](#).

Obtaining the Latest Version of the Ports Tree

If you are planning on running the latest version of the Ports Tree, you should have the latest release of OpenBSD. This is because of constant changing and fixing of the ports tree to help with its interaction with the rest of your OpenBSD system. If you don't follow this guideline, you can expect some minor problems while upgrading.

The best way to stay current with your OpenBSD Ports Tree is to use via `cvs(1)`. OpenBSD has a whole group of Anonymous CVS servers that are available to anyone. To read more about cvs look over <http://www.openbsd.com/anoncv.html> and [Faq8.5](#).

Though, if you do not have the ports tree at all, you can download it via any of OpenBSD's ftp servers. You can get a list at <http://www.openbsd.com/ftp.html>. From here download `/pub/OpenBSD/2.5/ports.tar.gz` and untar this in `/usr/ports`. For example:

```
ericj@oshibana> ftp ftp://ftp.openbsd.org/pub/OpenBSD/2.5/ports.tar.gz
ericj@oshibana> sudo cp ports.tar.gz /usr
ericj@oshibana> cd /usr; sudo tar xzf ports.tar.gz
```

Once this is done, you can use [cvs\(1\)](#) to update your Ports Tree. If you started with a base Ports Tree of 2.5 there will be a few changes that you will need to make. Most notably `/usr/share/mk`, which holds files used by [make\(1\)](#), will need to be updated. If you have the src tree on your system, you can update `/usr/share/mk` via [cvs\(1\)](#) and simply:

```
ericj@oshibana> cd /usr/src/share/mk; sudo make install
```

This will put the proper files into place. After this is done, you might also need the latest version of the `make(1)` binary. You will have to create this, if you don't have the src tree on your system you can grab it via `cvs(1)` and compile it. Example:

```
ericj@oshibana> export CVS_RSH=/usr/bin/ssh
ericj@oshibana> cvs -danoncv@any.anoncv.server:/cvs get src/usr.bin/make
cvs server: Updating src/usr.bin/make
U src/usr.bin/make/Makefile
U src/usr.bin/make/Makefile.boot
U src/usr.bin/make/arch.c
U src/usr.bin/make/bit.h
U src/usr.bin/make/buf.c
U src/usr.bin/make/buf.h
U src/usr.bin/make/compat.c
[SNIP]
ericj@oshibana> cd src/usr.bin/make; sudo make install
```

If you already have the src on your machine, simply update make and install it. For now, this should be all you need to do to have the ports tree working properly.

A snapshot of the ports tree is also created daily and can be downloaded from any of the [OpenBSD ftp servers](#) as `/pub/OpenBSD/snapshots/ports.tar.gz`.

What ports are available? and how do i find them?

There are two ways of searching for ports. The first way is to check <http://www.openbsd.com/portstat.html>. There you will find a list of what ports are currently available. The other way is to actually use the ports tree to search for keywords. To do this use make search key="searchkey". Here is an example of a search for 'samba':

```
ericj@oshibana> make search key="samba"
Port:      samba-2.0.5a
Path:      /usr/ports/net/samba
Info:      free SMB and CIFS client and server for UNIX
Maint:     ports@openbsd.org
Index:     net
B-deps:
R-deps:
Archs:     any

Port:      ADMsmb-0.2
Path:      /usr/ports/security/ADMsmb
Info:      Samba security scanner
Maint:     dugsong@monkey.org
Index:     security
B-deps:
R-deps:
Archs:     any
```

Installing Ports

Ports are set up to be EXTREMELY easy to make and install. Here is an example install for someone wanting to install the X11 program xfig. You'll notice the dependencies are automatically detected and completed:

First you need to cd to the dir of the program you want. If you are searching for a program, you can either update your locate database, or use the search function talked about below. Once you are in the dir of the program you want, you can just type make install. For example.

```
fenetyllin:/usr/ports/graphics/xfig# make install
===> Extracting for xfig-3.2.2
===> xfig-3.2.2 depends on shared library: jpeg.62. - /usr/local/lib/libjpeg.so.62.0
found
===> xfig-3.2.2 depends on shared library: Xaw3d.6. - not found
===> Verifying install for Xaw3d.6. in /usr/ports/x11/Xaw3d
>> Xaw3d-1.3.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch from ftp://crl.dec.com/pub/X11/contrib/widgets/Xaw3d/R6.1/.
Connected to crl.dec.com.
220 crl.dec.com FTP server (Digital UNIX Version 5.60) ready.
331 Guest login ok, send ident as password.
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Type set to I.
250 CWD command successful.
250 CWD command successful.
Retrieving pub/X11/contrib/widgets/Xaw3d/R6.1/Xaw3d-1.3.tar.gz
local: Xaw3d-1.3.tar.gz remote: Xaw3d-1.3.tar.gz
227 Entering Passive Mode (192,58,206,2,5,14)
150 Opening BINARY mode data connection for Xaw3d-1.3.tar.gz (0.0.0.0,0) (290277
bytes).
100% |*****| 283 KB 00:00 ETA
226 Transfer complete.
290277 bytes received in 101.09 seconds (2.80 KB/s)
```

```

221 Goodbye.
===> Extracting for Xaw3d-1.3
/bin/mkdir -p /usr/ports/x11/Xaw3d/work/xc/lib/Xaw3d/X11/Xaw3d
cd /usr/ports/x11/Xaw3d/work/xc/lib/Xaw3d/X11/Xaw3d; ln -sf ../../*.h .
===> Patching for Xaw3d-1.3
===> Configuring for Xaw3d-1.3
mv -f Makefile Makefile.bak
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
make Makefiles
[snip]

```

Listing Installed ports/packages

You can see a list of both ports and packages by using the `pkg_info` command.

```

bsd# /usr/sbin/pkg_info
zsh-3.0.5          The Z shell.
screen-3.7.4      A multi-screen window manager.
ssh-1.2.21        Secure shell client and server (remote login program).
emacs-20.2        GNU editing macros.
lynx-2.7.1ac-0.107 An alphanumeric display oriented World-Wide Web Client.
tcsh-6.07.02     An extended C-shell with many useful features.
bash-2.01         The GNU Borne Again Shell.
zip-2.2           Create/update ZIP files compatible with pkzip.
mm-2.7           Implementation of MIME, the Multipurpose Internet Mail Exten
ircii-2.8.2-epic3.004 An enhanced version of ircII, the Internet Relay Chat client
ispell-3.1.20     An interactive spelling checker.
tin-1.3.970930   TIN newsreader (termcap based)
procmail-3.11p7  A local mail delivery agent.
strobe-1.03      Fast scatter/gather TCP port scanner
lsof-4.15        Lists information about open files.
xntp3-5.92       Network Time Protocol Implementation.
ncftp-2.4.3      Network File System Client
nmh-0.27         The New MH mail handling program
bzip2-0.1p12    A block-sorting file compressor

```

Other Information

More information about the ports can be found in the [ports\(7\)](#) man page.

Our ports tree is constantly being expanded, and if you would like to help please see: <http://www.openbsd.com/ports.html>

8.7 - What are packages?

Packages are the precompiled binaries of some of the most used programs. They are ready for use on an OpenBSD system. Again, like the ports, packages are very easy to maintain and update. Packages are constantly being added so be sure to check each release for additional packages.

Here is a list of tools used in managing packages.

- [pkg_add\(1\)](#) - a utility for installing software package distributions
- [pkg_create\(1\)](#) - a utility for creating software package distributions
- [pkg_delete\(1\)](#) - a utility for deleting previously installed software package distributions
- [pkg_info\(1\)](#) - a utility for displaying information on software packages

Where to find packages

If you are a smart user and bought one of the [OpenBSD CD](#), then packages can be found on both CD's depending on your architecture. If you don't have an OpenBSD CD in your possession you can download packages from any of the ftp mirrors. You can get a list of mirrors <http://www.openbsd.com/ftp.html>. Packages are located at `/pub/OpenBSD/2.6/packages` from there packages are broken down depending on architecture.

Installing Packages

To install packages, the utility `pkg_add(1)` is used. `pkg_add(1)` is an extremely easy utility to use, in the following two examples `pkg_add(1)` will be used to install a package. The first example will show `pkg_add(1)` installing a package that resides on a local disk, the second example will show an installation of a package via ftp. In both examples `screen-3.7.6` will be installed.

Installing via local disk

```
$ sudo pkg_add -v screen-3.7.6.tgz
Requested space: 749864 bytes, free space: 2239117312 bytes in
/var/tmp/instmp.cpsHA27596
Running install with PRE-INSTALL for `screen-3.7.6'
extract: Package name is screen-3.7.6
extract: CWD to /usr/local
extract: /usr/local/bin/screen-3.7.6
extract: execute 'ln -sf screen-3.7.6 /usr/local/bin/screen'
extract: /usr/local/man/man1/screen.1
extract: /usr/local/info/screen.info
extract: execute '[ -f /usr/local/info/dir ] || sed -ne '1,/Menu:/p'
/usr/share/info/dir > /usr/local/info/dir'
extract: execute 'install-info /usr/local/info/screen.info /usr/local/info/dir'
extract: /usr/local/lib/screen/screencap
extract: /usr/local/lib/screen/screenrc
extract: CWD to .
Runningmtree for `screen-3.7.6'
mtree -q -U -f +MTREE_DIRS -d -e -p /usr/local
Running install with POST-INSTALL for `screen-3.7.6'
```

```
+-----
| The file /etc/screenrc has been created on your system.
| You may want to verify/edit its contents
|
| The file /usr/local/lib/screen/screencap contains a
| termcap like description of the screen virtual terminal.
| You may use it to update your terminal database.
| See termcap\(5\).
+-----
```

```
Attempting to record package into `/var/db/pkg/screen'
Package `screen-3.7.6' registered in `/var/db/pkg/screen-3.7.6'
```

In this example the `-v` flag was used to give a more verbose output, this option is not needed, but is helpful for debugging and was used here to give a little more insight into what `pkg_add(1)` is actually doing. Notice however, that there are some valid messages given out mentioning `/etc/screenrc`. Messages like this will be given to you whether or not you use the `-v` flag.

Installing via ftp

```
$ sudo pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/2.6/packages/i386/screen-3.7.6.tgz
>>> ftp -o - ftp://ftp.openbsd.org/pub/OpenBSD/2.6/packages/i386/screen-3.7.6.tgz
```

```
+-----
| The file /etc/screenrc has been created on your system.
| You may want to verify/edit its contents
```

```

| The file /usr/local/lib/screen/screencap contains a
| termcap like description of the screen virtual terminal.
| You may use it to update your terminal database.
| See termcap(5).
+-----

```

In this example you can see that I installed the i386 package, you should substitute this with your architecture. Notice: Not all architectures have the same packages. Some ports don't work on certain architectures. In this example the **-v** flag wasn't used, so only NEEDED messages are shown.

Viewing and Deleting Installed Packages

The utility [pkg_info\(1\)](#) is used to view a list of packages that are already installed on your system. This is usually needed to find out the correct name of a package before you remove that package. To see what packages are installed on your system simple use:

```

$ pkg_info
sslUSA26          ssl26.1 USA-only non-commercial crypto libs incl. SSL & RSA
mpg123-0.59q      mpeg audio 1/2 layer 1, 2 and 3 player
nmap-2.3b6        port scanning large networks
ircii-2.8.2-epic3.004enhanced version of ircII (internet relay chat)
screen-3.7.6      multi-screen window manager
unzip-5.40        extract, list & test files in a ZIP archive
xntp3-5.93e-export Network Time Protocol implementation
icb-5.0.9         Internet CB - mostly-defunct chat client

```

To delete a package, simple take the proper name of the package as shown by [pkg_info\(1\)](#) and use [pkg_delete\(1\)](#) to remove the package. In the below example, the screen package is being removed. Notice that on some occasions there are instructions of extra objects that need to be removed that [pkg_delete\(1\)](#) did not remove for you. As with the [pkg_add\(1\)](#) utility, you can use the **-v** flag to get more verbose output.

```

$ sudo pkg_delete screen-3.7.6

```

```

+-----
| To completely deinstall the screen-3.7.6 package you need to perform
| this step as root:
|
|         rm -f /etc/screenrc
|
| Do not do this if you plan on re-installing screen-3.7.6
| at some future time.
+-----

```

8.8 - Is there any way to use my floppy drive if it's not attached during boot?

Sure. You need to add "flags 0x20" at the end of the fd* entry and recompile your kernel. The line should be read:

```
fd*      at fdc? drive ? flags 0x20
```

After that you would be able to use the floppy drive all the times. It doesn't matter if you plugged it in after boot.

8.9 - Boot time Options - Using the OpenBSD bootloader

When booting your OpenBSD system, you have probably noticed the boot prompt.

```
boot>
```

For most people, you won't have to do anything here. It will automatically boot if no commands are given. But sometimes problems arise, or special functions are needed. That's where these options will come in handy. To start off, you should read through the

[boot\(8\)](#) man page. Here we will go over the most common used commands for the bootloader.

To start off, if no commands are issued, the bootloader will automatically try to boot **/bsd**. If that fails it will try **/obsd**, and so on till it finds a bootable kernel. You can specify this by hand by typing:

```
boot> boot wd0a:/bsd
```

or

```
boot> b /bsd
```

This will work if device wd0a is configured as your root device.

Here is a brief list of options you can use with the OpenBSD kernel.

- **-a** : This will allow you to specify an alternate root device after booting the kernel.
- **-c** : This allows you to enter the boot time configuration. Check the [Boot Time Config](#) section of the faq.
- **-s** : This is the option to boot into single user mode.
- **-d** : This option is used to dump the kernel into ddb. Keep in mind that you must have DDB built into the kernel.
- **-b** : Shortcut for RB_HALT.

These are entered in the format of: **boot [image [-abcd]]**

For further reading you can read [boot_i386\(8\)](#) man page

8.10 - S/Key

S/Key is a "one-time password" scheme. This allows for one-time passwords for use on un-secured channels. This can come very handy for those who don't have the ability to use ssh or any other encrypted channels. OpenBSD's S/Key implementation can use a variety of algorithms as the one-way hash. Here is the list of algorithms available:

- [md4](#)
- [md5](#)
- [sha1](#)
- [rmd160](#).

Setting up S/Key - The first steps

To start off the file `/etc/skeykeys` must exist. If this file is not in existence, have the super-user create it. This can be done simply by doing:

```
# touch /etc/skeykeys
```

Once that file is in existence, you can initialize your S/Key. To do this you will have to use [skeyinit\(1\)](#). With `skeyinit(1)`, you will first be prompted for your password to the system. This is the same password that you used to log into the system. Running `skeyinit(1)` over an insecure channel is completely not recommended, so this should be done over a secure channel (such as ssh) or the console. Once you have authorized yourself with your system password you will be asked for yet another password. This password is the *secret password*, and is **NOT** your system password. The secret password is not limited to 8 characters like system passwords, actually it must be at least 10 characters. A few word phrases are suggested. Here is an example user being added.

```
oshibana:ericj> skeyinit ericj
[Adding ericj]
Reminder - Only use this method if you are directly connected
           or have an encrypted channel.  If you are using telnet
           or rlogin, exit with no password and use skeyinit -s.
Enter secret password:
Again secret password:
```

```
ID ericj skey is otp-md5 99 oshi45820
Next login password: HAUL BUS JAKE DING HOT HOG
```

One line of particular importance in here is `ID ericj skey is otp-md5 99 oshi45820`. This gives a lot of information to the user. Here is

a breakdown of the sections and their importance.

- *otp-md5* - This shows which one-way was used to create your One-Time Password (otp).
- *99* - This is your sequence number. This is a number from 100 down to 1. Once it reaches one, another secret password must be created.
- *oshi45820* - This is your key.

But of more immediate importance is your password. Your password consists of 6 small words, combined together this is your password, spaces and all.

Actually using S/Key to login.

By now your skey has been initialized, and you have your password. You're ready to login. Here is an example session using s/key to login.

```

oshibana:ericj> ftp localhost
Connected to localhost.
220 oshibana.shin.ms FTP server (Version 6.4/OpenBSD) ready.
Name (localhost:ericj): ericj
331 Password [ otp-md5 96 oshi45820 ] for ericj required.
Password:
230- OpenBSD 2.5-current (OSHIBANA) #8: Tue Jun 22 19:20:16 EDT 1999
230-
230- Welcome to OpenBSD: The proactively secure Unix-like operating system.
230-
230- Please use the sendbug(1) utility to report bugs in the system.
230- Before reporting a bug, please try to reproduce it with the latest
230- version of the code. With bug reports, please try to ensure that
230- enough information to reproduce the problem is enclosed, and if a
230- known fix for it exists, include that as well.
230-
230 User ericj logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> quit
221 Goodbye.
```

Some of you might have noticed that my sequence number has changed. *otp-md5 96 oshi45820*. This is because by now I have used s/key to login several times. But how do you get your password after you've logged in once? Well to do this, you'll need to know what sequence number you're using and your key. As you're probably thinking, how can you remember which sequence number you're on? Well this is simple, use [skeyinfo\(1\)](#), and it will tell you what to use. For example here, I need to generate another password for a login that I might have to make in the future. (remember I'm doing this from a secure channel).

```

oshibana:ericj> skeyinfo
95 oshi45820
```

From this I can create the password for my next login. To do so, I'll use [skey\(1\)](#). I can use exactly that output from above to create my password.

```

oshibana:ericj> skey 95 oshi45820
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:
NOOK CHUB HOYT SAC DOLE FUME
```

I'm sure many of you won't always have a secure connect to create these passwords, and creating them over an insecure connection isn't feasible, so how can you create multiple passwords at one time? Well you can supply skey(1) with a number of how many passwords you want created. This can then be printed out and taken with you wherever you go.

```

oshibana:ericj> skey -n 5 95 oshi45820
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:
91: SHIM SET LEST HANS SMUG BOOT
```

8.0 - General Questions

```
92: SUE ARTY YAW SEED KURD BAND
93: JOEY SOOT PHI KYLE CURT REEK
94: WIRE BOGY MESS JUDE RUNT ADD
95: NOOK CHUB HOYT SAC DOLE FUME
```

Notice here though, that the bottom password should be the first used, because we are counting down from 100.

Using S/Key with telnet(1), ssh(1), and rlogin(1)

Using S/Key with telnet(1), ssh(1), or rlogin(1) is done in pretty much the same fashion as with ftp, only your first password must be "s/key". Example:

```
ericj@oshibana> telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

OpenBSD/i386 (oshibana) (ttyp2)
```

```
login: ericj
Password: <----- "s/key" entered.
otp-md5 98 oshi45821
Response: SCAN OLGA BING PUB REEL COCA
Last login: Thu Oct 7 12:21:48 on ttyp1 from 156.63.248.77
Warning: no Kerberos tickets issued.
OpenBSD 2.5-current (OShibANA) #4: Thu Sep 2 23:36:16 EDT 1999
```

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system. Before reporting a bug, please try to reproduce it with the latest version of the code. With bug reports, please try to ensure that enough information to reproduce the problem is enclosed, and if a known fix for it exists, include that as well.

```
You have mail.
ericj@oshibana>
```

Controlling S/Key

For more control over S/Key there is the `/etc/skey.access` file. (This does not exist by default, so it must be created.) This file can restrict S/Key in three primary ways.

- permit internet - Which will allow certain IPs to connect and use s/key.
- permit user - Which will allow users by login name access to s/key use.
- permit port - Which is really only useful for assigned ports, (ie dial-up)

If I wanted to allow one single user (ericj), from a certain IP (10.1.1.5), I would create a file like so.

```
# cat /etc/skey.access
permit internet 10.1.1.5
permit user ericj
```

8.11 - Why is my Macintosh losing so much time?

This is caused by a hardware bug. OpenBSD uses clock interrupts to keep track of the current time, but these interrupts have the lowest priority in Apple's architecture. So, under heavy load, (such as disk or network activity) clock interrupts will be lost and the Unix clock will not advance as it should.

MacOS gets around the time problem by always reading the hardware clock. OpenBSD only reads the hardware clock at boot time

and thereafter ignores it. You may notice that, at shutdown, the kernel is not confident enough to write the Unix time back into the hardware clock because this time loss problem is well known.

The best solution is to run `xntpd` (found in the ports collection) and just deal with the occasional lossage. Sometimes the lossage is so bad that even `xntpd` is afraid to skip the time. In this case, add the `-g` option to `ntpd` in `/etc/rc.securelevel` to force tracking.

Another simpler but less precise solution is to run `rdate(8)` on a regular basis, for example by having a crontab entry for it, preferably with the `-a` option so there is no "jump" in time. Another good place to launch [rdate\(8\)](#) is in your `/etc/ppp/ppp.linkup` file if you are not permanently connected and are a PPP user.

See also: <http://www.macbsd.com/macbsd/macbsd-docs/faq/faq-3.html#ss3.17>

8.12 - Will OpenBSD run on multiprocessor machines?

No. OpenBSD/i386 will not make use of multiple processors, but will run using one processor on a multi-processor system board. OpenBSD/sparc will not run on a multi-processor sparc system at all. No other platforms currently support multi-processor capabilities.

The reason is quite simple: there are not enough developers who have access to MP machines. If you want to donate MP hardware, please refer to <http://www.openbsd.com/donations.html>. Our developers are currently asking for i386, sparc, hppa, and other multi-processor hardware donations to begin SMP support. A project to bring multi-processor support to OpenBSD is underway, and progress can be check at <http://www.openbsd.com/smp.html>.

8.13 - I get Input/output error when trying to use my tty devices

As of OpenBSD 2.6, you need to use `/dev/cuaXX` for connections initiated from the OpenBSD system, the `/dev/ttyXX` devices are intended only for terminal or dial-in usage. While it was possible to use the tty devices in the past, the OpenBSD kernel is no longer compatible with this usage.

From [cua\(4\)](#):

For hardware terminal ports, dial-out is supported through matching device nodes called calling units. For instance, the terminal called `/dev/tty03` would have a matching calling unit called `/dev/cua03`. These two devices are normally differentiated by creating the calling unit device node with a minor number 128 greater than the dial-in device node. *Whereas the dial-in device (the tty) normally requires a hardware signal to indicate to the system that it is active, the dial-out device (the cua) does not, and hence can communicate unimpeded with a device such as a modem.* This means that a process like [getty\(8\)](#) will wait on a dial-in device until a connection is established. Meanwhile, a dial-out connection can be established on the dial-out device (for the very same hardware terminal port) without disturbing anything else on the system. The [getty\(8\)](#) process does not even notice that anything is happening on the terminal port. If a connecting call comes in after the dial-out connection has finished, the [getty\(8\)](#) process will deal with it properly, without having noticed the intervening dial-out action.

8.14 - Where can I find a Netscape binary for OpenBSD?

There is no OpenBSD Netscape binary. However since OpenBSD supports binary emulation of many other operating systems we are able to use other binaries.

- For I386 systems, it is best to use the BSDI binary.
- For SPARC systems, it is best to use the SUN-OS binary.

Using the [Ports System](#) you can install these with no hassle. The Navigator Port and Communicator Ports are located at `/usr/ports/www/netscape` Once you have the ports tree installed.

8.15 - How do I use the mg editor which is now part of OpenBSD (post-2.6!!)?

Mg is a micro Emacs-style text editor. Micro means that it's small (Emacs is very large!) For the basics, read the [mg\(1\)](#) manual page

and the [tutorial](#), as included with the source code. For more interesting questions (such as, "I don't have a Meta key!") check out the [Emacs FAQ](#).

Note that since mg is a small Emacs implementation, which is mostly similar to the text editor features of Emacs 17, it does not implement many of Emacs' other functionality. (Including mail and news functionality, as well as modes for Lisp, C++, Lex, Awk, Java, etc...)

8.16 - Ksh does not appear to read my .profile!

There are two possible reasons for this.

- .profile is not owned by the user This is easy to fix. For **username**,
chown username ~username/.profile
- You are using ksh from within X windows

Under xterm, argv[0] for ksh is not prepended with a dash. Prepending - to argv[0] will cause csh and ksh to know they should interpret their login files. (For csh that's .login, with a separate .cshrc that is always run when csh starts up. With ksh, this is more noticeable because there is only one startup script, .profile. This file is ignored unless the shell is a login shell.)

To fix this, create a file in your home directory called .Xdefaults with the phrase *XTerm*loginShell: true*

```
$ echo "XTerm*loginShell: true" > ~/.Xdefaults
```

You may not have had to do this before, because some installations of X Windows come with this setting as default. OpenBSD has chosen to follow the XFree86 behavior.

8.17 - Why does my /etc/motd file get written over when I modified it?

The */etc/motd* file is edited upon every boot of the system, replacing the first line with the systems kernel version information. When editing this file, make sure that you start 2 lines from the top, to keep */etc/rc* from deleting these lines when it edits */etc/motd* upon boot.

8.18 - Why does www.openbsd.org run on Solaris?

Although none of the developers think it is particularly relevant, this question comes up frequently enough in the mailing lists that it is answered here. *www.openbsd.org* and the main OpenBSD ftp site are hosted at a SunSITE at the University of Alberta, Canada. These sites are hosted on a large Sun system, which has access to lots of storage space and Internet bandwidth. The presence of the SunSITE gives the OpenBSD group access to this bandwidth. This is why the main site runs here. Many of the OpenBSD mirror sites run OpenBSD, but since they do not have guaranteed access to this large amount of bandwidth, the group has chosen to run the main site at the University of Alberta SunSITE.

[\[Back to Main Index\]](#) [\[To Section 7.0 - Keyboard controls\]](#) [\[To Section 9.0 - Tips for linux users\]](#)



www@openbsd.org

\$OpenBSD: faq8.html,v 1.54 2000/06/09 21:42:40 chris Exp \$

OpenBSD

9.0 - Migrating from Linux

Table of Contents

- [9.1 - Tips for Linux \(and other free Unix-like OS\) users](#)
 - [9.2 - Dual boot of Linux and OpenBSD](#)
 - [9.3 - Converting your linux \(or other System-7 style\) password file to BSD-style.](#)
 - [9.4 - Getting OpenBSD and Linux to interact](#)
-

9.1 - Simple tips for Linux (and other free Unix-like OS) users

There are several differences between OpenBSD and Linux. These differences include but are not limited to, bootup procedure, network interface usage and disk management. Most differences are well documented, but involve searching manpages. This document tries to be an index of those differences.

- OpenBSD has a [ports tree](#). This is to accomodate the fact that at this point not many applications are native to the OpenBSD environment. This is both an attempt to get applications to work on OpenBSD for end-users and to get more applications made with OpenBSD in mind. Eventually this ports tree will be used to make a nice set of binary packages.
- OpenBSD uses CVS for source changes. With Linux, source code is disseminated through separate distributions. OpenBSD has pioneered anonymous CVS, which allows anyone to extract the full source tree for any version of OpenBSD (from 2.0 to current, and all revisions of all files in between) at any time! There is also a very convenient and easy to use [web interface to CVS](#).
- OpenBSD periodically releases snapshots for various architectures and makes a stable, official CD release every 6 months.
- OpenBSD contains STRONG CRYPTO, which USA based OS's can't contain. (See <http://www.openbsd.com/crypto.html>) OpenBSD has also gone through heavy security auditing and many security features have already been implemented into the source tree. (IPSEC, KERBEROS).
- OpenBSD's kernel is /bsd.
- The names of hard disks are usually /dev/wd and /dev/sd (ATA/SCSI)
- [/sbin/ifconfig](#) with no arguments in Linux gives the state of all the interfaces. Under OpenBSD you need the -a flag.
- [/sbin/route](#) with no arguments in Linux gives the state of all the active routes. Under OpenBSD you need the "show" parameter, or do a **netstat -r** (nice).
- OpenBSD comes with Darren Reed's IP Filter package, not ipfw. This means that:
 - IP-Masquerading is done through ipnat. ([ipnat\(1\)](#))
 - ipfwadm is done through ipf ([ipf\(1\)](#), [ipf\(5\)](#))
 - You should look at [section 6](#) for detailed configuration assistance and information.
- Interface address is stored in [/etc/hostname.<interfacename>](#). It can be a name instead of an IP address.

- The machine name is in `/etc/myname`
- The default gateway is in `/etc/mygate`
- The network interface aliases are in `/etc/ifaliases`. **NOTE:** This has been phased out after 2.6. Starting in OpenBSD 2.7, this will be setup in `/etc/hostname.if`.
- OpenBSD's default shell is `/bin/sh`, which is the Korn shell. Shells such as `bash` and `tcsh` can be added as packages or installed from the ports tree.
- Password management changes a lot. The main files are different. ([passwd\(1\)](#), [passwd\(5\)](#))
- Devices are named by driver, not by type. So for example, there are no `eth*` devices. It would be `ne0` for an ne2000 ethernet card, and `xl0` for a 3Com Etherlink XL and Fast Etherlink XL ethernet device, etc.
- OpenBSD developers have made serious efforts to keep the manual pages up-to-date and accurate. Use the [man\(1\)](#) command to find information.

9.2 - Dual booting Linux and OpenBSD

Yes! it is possible!

Read [INSTALL.linux](#)

9.3 - Converting your Linux (or other System 7-style) password file to BSD-style

First, figure out if your Linux password file is shadowed or not. If it is, grab [John the Ripper](#) and use the `unshadow` utility that comes with it to merge your `passwd` and `shadow` files into one System 7-style file.

Using your Linux password file, we'll call it `linux_passwd`, you need to add in `::0:0` between fields four and seven. `Awk` does this for you.

```
# cat linux_passwd | awk -F :  
'{printf("%s:%s:%s:%s::0:0:%s:%s:%s\n",$1,$2,$3,$4,$5,$6,$7); }' > new_passwd
```

At this point, you want to edit the `new_passwd` file and remove the root and other system entries that are already present in your OpenBSD password file or aren't applicable with OpenBSD (all of them). Also, make sure there are no duplicate usernames or user IDs between `new_passwd` and your OpenBSD box's `/etc/passwd`. The easiest way to do this is to start with a fresh `/etc/passwd`.

```
# cat new_passwd >> /etc/master.passwd  
# pwd_mkdb -p /etc/master.passwd
```

The last step, `pwd_mkdb` is necessary to rebuild the `/etc/spwd.db` and `/etc/pwd.db` files. It also creates a System 7-style password file (minus encrypted passwords) at `/etc/passwd` for programs which use it. OpenBSD uses a stronger encryption for passwords, blowfish, which is very unlikely to be found on any system which uses full System 7-style password files. To switch over to this stronger encryption, simply have the users run `'passwd'` and change their password. The new password they enter will be encrypted with your default setting (usually blowfish unless you've edited `/etc/passwd.conf`). Or, as root, you can run `passwd username`.

9.4 - Getting OpenBSD and Linux to interact

If you are migrating from Linux to OpenBSD, note that OpenBSD has `COMPAT_LINUX` enabled by default in the `GENERIC` kernel. To run any Linux binaries that are not statically linked (most of them), you need to follow the instructions on the [compat_linux\(8\)](#) manual page. A simple way to get most of the useful Linux libraries is to install

linux_lib from your ports collection. To find out more about the Ports collection read [FAQ 8.6](#). If you already have the ports tree installed use these commands to get linux libraries installed.

```
# cd /usr/ports/emulators/linux_lib
# make install
```

OpenBSD supports the EXT2FS file system. Use **disklabel *disk*** (where *disk* is the device name for your disk.) to see what OpenBSD thinks your Linux partition is (but **don't** use disklabel or fdisk to make any changes to it). For further information on using disklabel read [FAQ 14.1](#).

[\[Back to Main Index\]](#) [\[To Section 8.0 - General Questions\]](#) [\[To Section 10.0 - System Administration\]](#)



www@openbsd.org

\$OpenBSD: faq9.html,v 1.20 2000/04/07 19:23:08 ericj Exp \$

10.0 - System Management

Table of Contents

- [10.1 - When I try to su to root it says that I'm in the wrong group](#)
 - [10.2 - How do I duplicate a filesystem?](#)
 - [10.3 - How do I start daemons with the system? \(Overview of rc\(8\) \)](#)
 - [10.4 - Why do users get relaying access denied when they are remotely sending mail through my OpenBSD system?](#)
 - [10.5 - I've set up POP, but I get errors when accessing my mail through POP. What can i do?](#)
 - [10.6 - Setting up a Secure HTTP Server using SSL\(8\)](#)
 - [10.7 - I made changes to /etc/passwd with vi\(1\), but the changes didn't seem to take place. Why?](#)
 - [10.8 - How do I add a user? or delete a user?](#)
 - [10.9 - How do I create a ftp-only account?](#)
 - [10.10 - Setting up user disk quotas](#)
 - [10.11 - Setting up Kerberos Client/Server](#)
 - [10.12 - Setting up an Anonymous FTP Server](#)
 - [10.13 - Confining users to their home dir's in ftpd\(8\).](#)
-

10.1 - Why does it say that I'm in the wrong group when I try to su root?

Existing users must be added to the "wheel" group by hand. This is done for security reasons, and you should be cautious with whom you give access to. On OpenBSD, users who are in the wheel group are allowed to use the [su\(1\)](#) userland program to become root. Users who are not in "wheel" cannot use su(1). Here is an example of a /etc/group entry to place the user **ericj** into the "wheel" group.

If you are adding a new user with [adduser\(8\)](#), you can put them in the wheel group by answering wheel at Invite *user* into other groups: This will add them to /etc/group, which will look something like this:

```
wheel:*:0:root,ericj
```

If you are looking for a way to allow users limited access to superuser privileges, without putting them in the "wheel" group, use [sudo\(8\)](#).

10.2 - How do I duplicate a filesystem?

To duplicate your filesystem use [dump\(8\)](#) and [restore\(8\)](#). For example. To duplicate everything under directory SRC to directory DST, do a:

```
# cd /SRC; dump 0f - . | (cd /DST; restore -rf - )
```

dump is designed to give you plenty of backup capabilities, and it may be an overkill if you just want to duplicate a part of a (or an entire) filesystem. The command [tar\(1\)](#) may be faster for this operation. The format looks very similar:

```
# cd /SRC; tar cf - . | (cd /DST; tar xpf - )
```

11.0 - Performance Tuning

Table of Contents

- [11.1 - Networking](#)
 - [11.2 - Disk I/O](#)
 - [11.3 - Tuning kmem](#)
 - [11.4 - Hardware Choices](#)
 - [11.5 - Why aren't we using async mounts?](#)
-

11.1 - Networking

If you run a busy server, gateway or firewall, you should make sure to prevent memory starvation to various parts of the kernel described below.

The [options\(4\)](#) man page talks about the options presented.

An option you may need to change for a busy server, gateway or firewall is NMBCLUSTERS. This controls the size of the kernel mbuf cluster map. On your computer, if you get messages like "mb_map full", you need to increase this value. If traffic on a networking interface stops for no apparent reason, this may also be a sign that you need to increase this value. A reasonable value on the i386 port with most 100Mbps ethernet interfaces (no matter how many the machine has) is 8192.

option NMBCLUSTERS=8192

11.2 - Disk I/O

Disk I/O speed is a significant factor in the overall speed of your computer. It becomes increasingly important when your computer is hosting a multi-user environment (users of all kinds, from those who log-in interactively to those who see you as a file-server or a web-server.) Data storage constantly needs attention, especially when your partitions run out of space and when your disks fail. OpenBSD has several options to increase the speed of your disk operations and provide fault tolerance.

CCD

The first option is the use of [ccd\(4\)](#), the Concatenated Disk Driver. This allows you to join several partitions into one virtual disk (and thus, you can make several disks look like one disk). This concept is similar to that of LVM (logical volume management), which is found in many commercial Unix flavors.

If you are running GENERIC, ccd is already enabled. If not, you may need to add it to your kernel configuration. To start the setup of ccd, you need to add support for it in your kernel. A line such as:

```
pseudo-device    ccd        4          # concatenated disk devices
```

The above example gives you up to 4 ccd devices (virtual disks). Now you need to figure out what partitions on your real disks that you want to dedicate to ccd. Use `disklabel` to mark these partitions as type 'ccd'. On some architectures, `disklabel` may not allow you to do this. In this case, mark them as 'ffs'.

If you are using ccd to gain performance by striping, note that you will not get optimum performance unless you use the same model of disks with the same `disklabel` settings.

Edit `/etc/ccd.conf` to look something like this: (for more information on configuring ccd, look at [ccdconfig\(8\)](#))

```
# Configuration file for concatenated disk devices
#
# ccd   ileave  flags   component devices
ccd0   16      none   /dev/sd2e /dev/sd3e
```

To make your changes take effect, run

```
# ccdconfig -C
```

As long as `/etc/ccd.conf` exists, ccd will automatically configure itself upon boot. Now, you have a new disk, `ccd0`, a combination of `/dev/sd2e` and `/dev/sd3e`. Just use `disklabel` on it like you normally would to make the partition or partitions you want to use. Again, don't use the 'c' partition as an actual partition that you put stuff on. Make sure your useable partitions are at least one cylinder off from the beginning of the disk.

RAID

Another solution is [raid\(4\)](#) which will have you use [raidctl\(8\)](#) to control your raid devices. OpenBSD's RAID is based upon Greg Oster's [NetBSD port](#) of the CMU [RAIDframe](#) software. OpenBSD has support for RAID levels of 0, 1, 4, and 5.

With `raid`, as with `ccd`, support must be in the KERNEL. Unlike `ccd`, support for RAID is not found in `GENERIC`, it must be compiled into your kernel (RAID support adds some 500K to the size of an i386 kernel!)

```
pseudo-device   raid   4           # RAIDframe disk device
```

Setting up RAID on some operating systems is confusing and painful to say the least. Not so with `RAIDframe`. Read the [raid\(4\)](#) and [raidctl\(8\)](#) man pages to get full details. There are many options and possible configurations, a detailed explanation is beyond the scope of this document.

Filesystem Buffer

For file servers with memory to spare, you can increase `BUFCACHEPERCENT`. That is, what percentage of your RAM should you use as a file system buffer. This option may change when the Unified Buffer Cache is completed and is part of OpenBSD. In the mean time, to increase `BUFCACHEPERCENT`, you should add a line to your kernel configuration like this:

```
option BUFCACHEPERCENT=30
```

Of course you can make it as low as 5 percent (the default) or as high as 50 percent (or more.)

Soft updates

Another tool that can be used to speed up your system is `softupdates`. One of the slowest operations in the traditional BSD file system is updating metainfo (which happens, among other times, when you create or delete files and directories.) `Softupdates` attempts to update metainfo in RAM instead of writing to the hard disk each and every single metainfo update. Another effect of this is that the metainfo on disk should always be complete, although not always up to date. So, a system crash should not require `fsck` upon boot up, but simply a background version of `fsck` that makes changes to the metainfo in RAM (a la `softupdates`). This means rebooting a server is much faster, because you don't have to wait for `fsck`! (OpenBSD does not have this feature yet.) You can read more about `softupdates` in the [softupdates FAQ](#) entry.

11.3 - Tuning `kmem`

This was moved to the [OpenBSD 2.5 information section](#).

11.4 - Hardware choices

(Note- this section is heavily centered around the i386, or PC, architecture. That is to say... other architectures don't give you quite as many choices!)

The performance of your applications depends heavily on your OS and the facilities it provides. This may be part of the reason that you are using OpenBSD. The performance of your applications also depends heavily on your hardware. For many folks, the Price/Performance ratio of a brand new PC with a Intel Pentium III or AMD Athlon processor is much better then the Price/Performance ratio of a Sun UltraSparc 60! Of course, the price of OpenBSD can't be beat.

If you are shopping for a new PC, whether you are buying it piece by piece or completely pre-built, you want to make sure first that you are buying reliable parts. In the PC world, this is not easy. **Bad or otherwise unreliable or mismatched parts can make OpenBSD run poorly and crash often.** The best advice we can give is to be careful, buy brands and parts that have been reviewed by an authority you trust. Sometimes, when you skimp on the price of a PC, you lose in quality!

There are certain things that will help bring out the maximum performance of your hardware:

- Use multiple disks.

Instead of buying one 20GB disk, buy multiple 9GB disks. While this may cost more, distributing the load over multiple spindles will decrease the amount of time necessary to access data on the disks. And, with more spindles, you will get more reliability and faster data access with RAID.

- Use SCSI if you need very high disk IO speeds.

IDE disks normally run at 5400 RPM, with new disks emerging from manufacturers that run at 7200 RPM. Using high end IDE disks, it may be unreasonable to expect more then 15 to 20 megabytes per second of throughput from a single disk. Using high end SCSI disks (higher cost 10000 RPM disks), you can achieve performance higher then this. Conversely, if you are using medium or low end SCSI disks, this is a waste of money, and IDE will serve you just as well, if not better.

If you are building a server, and you need more then 20GB of disk space, you may want to consider SCSI. IDE limits you to two disks per controller. Concurrent access to these two disks may have a negative impact on the I/O performance of these disks. Wide SCSI limits you to 15 per controller! While SCSI costs more, the

flexibility and performance can justify these costs in some environments.

- Use SDRAM instead of DRAM.

This option applies mainly to PCs. Most other architectures don't give you a choice of what kind of RAM you can use. Several PCs still do. You will get better performance with SDRAM versus DRAM (SIMMs). If your system supports RDRAM or some other new type of RAM, then you are already one step ahead...

- Use ECC or parity RAM.

Parity adds some functionality to see if the data in RAM has been corrupted. ECC extends this functionality and attempts to correct some bit corruption errors on the fly. This option applies mainly to PCs. Most other architectures simply require parity or ECC capable RAM. Several non-PC computers won't even boot with non-parity RAM. If you aren't using ECC/parity RAM, you may get data corruption and other abnormalities. Several manufacturers of "cheap PC RAM" don't even make an ECC variety! This will help you avoid them! PC manufacturers often sell several product lines, divided around "servers" and "workstations." The servers will incorporate ECC RAM into their architecture. Unix workstation manufacturers have been using parity (and now ECC) for several years in all of their product lines.

- Avoid ISA devices.

While most folks avoid ISA devices, because they are generally hard to configure and out of date, there are still plenty in existence. If you are using the ISA bus for your disk or network controllers, (or even worse, for both) remember that the ISA bus itself can be a performance bottleneck. If you need speed, look to PCI. Of course, there are still several ISA bus cards that work just fine. Unfortunately, most of these are sound cards and serial port cards.

- Avoid cheap PCI network adapters.

OpenBSD supports a plethora of cheap PCI network adapters. These adapters work great in home systems, and also low or moderate throughput business and research environments. But, if you need high throughput and low impact on your server, you are better off buying a quality PCI network adapter. Unfortunately, some expensive brand adapters (such as the 3com XL series) are not much better than the cheap adapters. One favorite 10/100Mbps adapter is the Intel EtherExpress PRO/100.

11.5 - Why aren't we using async mounts?

Question: "I simply do "mount -u -o async /" which makes one package I use (which insists on touching a few hundred things from time to time) useable. Why is async mounting frowned upon and not on by default (as it is in some other unixen) ? Surely it is much simpler and therefore a safer way of improving performance in some applications ?"

Answer: "Async mounts is indeed faster than sync mounts, but they are also less safe. What happens in case of a power failure? Or a hardware problem? The quest for speed should not sacrifice the reliability and the stability of the system. Check the manpage for [mount\(8\)](#)."

```

async    All I/O to the file system should be done asynchronously.
         This is a dangerous flag to set since it does not guarantee
         to keep a consistent file system structure on the disk. You
         should not use this flag unless you are prepared to recreate
         the file system should your system crash. The most common
         use of this flag is to speed up restore(8) where it can give
         a factor of two speed increase.
```

On the other hand, when you are dealing with temp data that you can recreate from scratch after a crash, you could gain speed by using a separate partition, used for that data only, mounted async. If you don't mind risking the loss of all the data in the partition when something goes wrong...

[\[to Section 12.0 - For Advanced Users\]](#) [\[To Section 10.0 - System Administration\]](#)



www@openbsd.org

\$OpenBSD: faq11.html,v 1.8 2000/03/03 13:17:15 rohee Exp \$

10.3 - How do I start daemons with the system? (Overview of rc(8))

OpenBSD uses an [rc\(8\)](#) style startup. This uses a few key files for startup.

- `/etc/rc` - Main script. Should not be edited.
- `/etc/rc.conf` - Configuration file used by `/etc/rc` to know what daemons should start with the system.
- `/etc/netstart` - Script used to initialize the network. Shouldn't be edited.
- `/etc/rc.local` - Script used for local administration. This is where new daemons or host specific information should be stored.
- `/etc/rc.securelevel` - Script which runs commands that must be run before the security level changes. See [init\(8\)](#)
- `/etc/rc.shutdown` - Script run on shutdown. Put anything you want done before shutdown in this file. See [rc.shutdown\(8\)](#)

How does rc(8) work?

The main files a system administrator should concentrate on are `/etc/rc.conf`, `/etc/rc.local` and `/etc/rc.shutdown`. To get a look of how the `rc(8)` procedure works, here is a the flow:

After the kernel is booted. `/etc/rc` is started.

- Filesystems checked. This will always be bypassed if the file `/etc/fastboot` exists. This is certainly not a good idea though.
- Configuration Variables are read in from `/etc/rc.conf`
- Filesystems are mounted
- Clears out `/tmp` and preserves any editor files
- Configures the network via `/etc/netstart`
 - Configures your interfaces up.
 - Sets your hostname, domainname, etc.
- Starts system daemons
- Does various checks. (quota's, savecore, etc)
- Local daemons are run, ala `/etc/rc.local`

Starting Daemons and Services that come with OpenBSD

Most daemons and services that come with OpenBSD by default can be started on boot by simply editing the `/etc/rc.conf` configuration file. To start out take a look at the default [/etc/rc.conf](#) file. You'll see lines similar to this:

```
ftpd_flags=NO                # for non-inetd use: ftpd_flags="-D"
```

A line like this shows that `ftpd` is not starting up with the system. (At least not via `rc(8)`, read the [Anonymous FTP FAQ](#) to read more about this.) In any case, each line also has a comment showing you the flags for **NORMAL** usage of that daemon or service. This doesn't mean that you must run that daemon or service with those flags. You can always use `man(1)` to see how you can have that daemon or service start up in any way you like. For example, Here is the default line pertaining to `httpd(8)`.

```
httpd_flags=NO                # for normal use: "" (or "-DSSL" after reading ssl(8))
```

Here you can obviously see that starting up `httpd` normally no flags are necessary. So a line like: "`httpd_flags=""`" would be necessary. But to start `httpd` with `ssl` enabled. (Refer to the [SSL FAQ](#) or [ssl\(8\)](#)) You should start with a line like: "`httpd_flags="-DSSL"`".

Starting up local daemons and configuration

For other daemons that you might install with the system via ports or other ways, you will use the `/etc/rc.local` file. For example, I've installed a daemon in which lie's at `/usr/local/sbin/daemonx`. I want this to start at boot time. I would put an entry into `/etc/rc.local` like this:

```
if [ -x /usr/local/sbin/daemonx ]; then
    echo -n ' daemonx';          /usr/local/sbin/daemonx
fi
```

From now on, this daemon will be run at boot. You will be able to see any errors on boot, a normal boot with no errors would show a line like this:

Starting local daemons: daemonx.

rc.shutdown

/etc/rc.shutdown is a script that is run a shutdown. Anything you want done before the system shuts down should be added to this file. If you have apm, you can also set "powerdown=YES". Which will give you the equivalent of "shutdown -p".

10.4 - Why do users get relaying access denied when they are remotely sending mail through my OpenBSD system?

Try this:

```
# cat /etc/mail/sendmail.cf | grep relay-domains
```

The output may look something like this:

```
FR-o /etc/mail/relay-domains
```

If this file doesn't exist, create it. You will need to enter the hosts who are sending mail remotely with the following syntax:

```
.domain.com      #Allow relaying for/to any host in domain.com
sub.domain.com   #Allow relaying for/to sub.domain.com and any host in that domain
10.2             #Allow relaying from all hosts in the IP net 10.2.*.*
```

Don't forget send a 'HangUP' signal to sendmail, (a signal which causes most daemons to re-read their configuration file):

```
# kill -HUP `cat /var/run/sendmail.pid`
```

Further Reading

- <http://www.sendmail.org/~ca/email/relayingdenied.html>
- <http://www.sendmail.org/tips/relaying.html>
- <http://www.sendmail.org/antispam.html>

10.5 - I've set up POP, but users have trouble accessing mail through POP. What can I do?

Most issues dealing with POP are problems with temporary files and lock files. If your pop server sends an error message such as:

```
-ERR Couldn't open temporary file, do you own it?
```

Try setting up your permissions as such:

```
permission in /var
drwxrwxr-x  2 bin      mail      512 May 26 20:08 mail
```

```
permissions in /var/mail
-rw-----  1 username  username  0 May 26 20:08 username
```

Another thing to check is that the user actually owns their own */var/mail* file. Of course this should be the case (as in, */var/mail/joe* should be owned by *joe*) but if it isn't set correctly it could be the problem!

Of course, making */var/mail* writeable by group *mail* opens up some vague and obscure security problems. It is likely that you will never have problems with it. But it could (especially if you are a high profile site, ISP,...)! Try running *cucipop* or another POP daemon from the OpenBSD ports collection. Or, you could just have the wrong options selected for your pop daemon (like dot locking). Or, you may just need to change the directory that it locks in (although then the locking would only be valueable for the POP daemon.)

10.6 - Setting up a Secure HTTP server with SSL(8)

Starting with OpenBSD 2.6, OpenBSD ships with an SSL-ready httpd. However, because of various patents, some libraries cannot be shipped. So, to be able to use RSA certificates you must install extra libraries onto your system. There are two sets of these libraries, depending on your location in the world. (**NOTE:** Those who did an FTP install, or any network install most likely have already installed this onto your system). For people living outside of the United States, you should use a package called "**ssl26.tgz**", Those living in the United States must use "**sslUSA26.tgz**". These are packages, therefore installed using the [pkg_add\(1\)](#) utility. For example, To install the NON-U.S. Resident version of the libraries use the following line where `${arch}` is your architecture. (**NOTE:** This is not available for the Alpha architecture due to its lack of shared libs.)

```
# pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/2.6/packages/${arch}/ssl26.tgz
```

For use with [httpd\(8\)](#), you must first have a certificate created. This will be kept in `/etc/ssl/private/`. The steps shown here are taken in part from the [ssl\(8\)](#) man page. Refer to it for further information. This FAQ entry only outlines how to create an RSA certificate for web servers, not a DSA server certificate. To find out how to do so, please refer to the [ssl\(8\)](#) man page.

To start off, you need to create your server key and certificate. To use the RSA features, you must have upgraded your libssl. Now you can create your key. Using OpenSSL:

```
# openssl genrsa -out /etc/ssl/private/server.key 1024
```

Or, if you wish the key to be encrypted with a passphrase that you will have to type in when starting servers

```
# openssl genrsa -des3 -out /etc/ssl/private/server.key 1024
```

The next step is to generate a Certificate Signing Request which is used to get a Certifying Authority (CA) to sign your certificate. To do this use the command:

```
# openssl req -new -key /etc/ssl/private/server.key -out /etc/ssl/private/server.csr
```

This `server.csr` file can then be given to Certifying Authority who will sign the key. One such CA is **Thawte Certification** which you can reach at <http://www.thawte.com/>. Thawte can currently sign RSA keys for you. A procedure is being worked out to allow for DSA keys.

If you cannot afford this, or just want to sign the certificate yourself, you can use the following.

```
# openssl x509 -req -days 365 -in /etc/ssl/private/server.csr \
    -signkey /etc/ssl/private/server.key -out /etc/ssl/server.crt
```

With `/etc/ssl/server.crt` and `/etc/ssl/private/server.key` in place, you should be able to start [httpd\(8\)](#) with the `-DSSL`

10.7 - I edited /etc/passwd, but the changes didn't seem to take place. Why?

If you edit `/etc/passwd`, your changes will be lost. OpenBSD generates `/etc/passwd` dynamically with [pwd_mkdb\(8\)](#). The main password file in OpenBSD is `/etc/master.passwd`. According to [pwd_mkdb\(8\)](#),

```
FILES
    /etc/master.passwd  current password file
    /etc/passwd         a Version 7 format password file
    /etc/pwd.db         insecure password database file
    /etc/pwd.db.tmp     temporary file
    /etc/spwd.db        secure password database file
    /etc/spwd.db.tmp   temporary file
```

In a traditional Unix password file, such as `/etc/passwd`, everything including the user's encrypted password is available to anyone on the system (and is a prime target for programs such as Crack.) 4.4BSD introduces the `master.passwd` file, which has an extended format (with additional options beyond what was provided by `/etc/passwd`) and is only readable by root. For faster access to data, the library calls which access this data normally read `/etc/pwd.db` and `/etc/spwd.db`.

OpenBSD does come with a tool with which you should edit your password file. It is called [vipw\(8\)](#). Vipw will use `vi` (or your favourite editor defined per `$EDITOR`) to edit `/etc/master.passwd`. After you are done editing, it will re-create `/etc/passwd`, `/etc/pwd.db`, and `/etc/spwd.db` as per your changes. Vipw also takes care of locking these files, so that if anyone else attempts to change them at the same time, they will be denied access.

10.8 - What is the best way to add and delete users?

For OpenBSD 2.6 users, the [adduser\(8\)](#) command is available for adding users. In OpenBSD 2.7, the [user\(8\)](#) command was added to deal with adding and removing both users and groups.

- [adduser\(8\)](#)
- [user\(8\)](#)

The best way to add a user in OpenBSD is to use the [adduser\(8\)](#) script. You can configure this to work however you like by editing `/etc/adduser.conf`. You can add users by hand via [vipw\(8\)](#), but this is the recommended way to add users. `adduser(8)` allows for consistency checks on `/etc/passwd`, `/etc/group`, and shell databases. It will create the entries and `$HOME` directories for you. It can even send a message to the user welcoming them. This can be changed to meet your needs. For further instructions on adding users read the [adduser_proc\(8\)](#) man page. Here is an example user, `testuser` being added to a system. His/Her `$HOME` directory will be placed in `/home/testuser`, and given the group `guest`, and the shell `/bin/ksh`.

```
# adduser
Use option ``-silent'' if you don't want to see all warnings and questions.
```

```
Reading /etc/shells
Check /etc/master.passwd
Check /etc/group
```

Ok, let's go.

Don't worry about mistakes. I will give you the chance later to correct any input.

```
Enter username [a-z0-9_]: testuser
```

```
Enter full name []: Test FAQ User
```

```
Enter shell csh ksh nologin sh [ksh]: ksh
```

```
Uid [1002]: <Enter>
```

```
Login group testuser [testuser]: guest
```

```
Login group is ``guest''. Invite testuser into other groups: guest no
```

```
[no]: no
```

```
Enter password []:
```

```
Enter password again []:
```

```
Name:      testuser
Password:  ****
Fullname:  Test FAQ User
Uid:       1002
Gid:       31 (guest)
Groups:    guest
HOME:      /home/testuser
Shell:     /bin/ksh
OK? (y/n) [y]: y
Added user ``testuser''
Copy files from /usr/share/skel to /home/testuser
Add another user? (y/n) [y]: n
Goodbye!
```

To delete users you should use the [rmuser\(8\)](#) utility. This will remove all existence of a user. It will remove any [crontab\(1\)](#) entries, their `$HOME` dir (if it is owned by the user), and their mail. Of course it will also remove their `/etc/passwd` and `/etc/group` entries. Next is an example of removing the user that was added above. Notice you are prompted for the name, and whether or not to remove the users home directory.

```
# rmuser
Enter login name for user to remove: testuser
Matching password entry:
```

```
testuser:$2a$07$ZWnBosbqMJ.ducQBfsTKUe3PL97Ve1AHWJ0A4uLamniLNXLeyrEie:1002:31::0:0:Test
FAQ User:/home/testuser:/bin/ksh
```

```

Is this the entry you wish to remove? y
Remove user's home directory (/home/testuser)? y
Updating password file, updating databases, done.
Updating group file: done.
Removing user's home directory (/home/testuser): done.

```

10.8.2 - Adding users via user(8)

Starting in the 2.7 release of OpenBSD, many commands were added for dealing with users and groups. These tools are less interactive than the [adduser\(8\)](#) command, which helps facilitate using these in a script.

A list of the added commands are:

- [group\(8\)](#)
- [groupadd\(8\)](#)
- [groupdel\(8\)](#)
- [groupinfo\(8\)](#)
- [groupmod\(8\)](#)
- [user\(8\)](#)
- [useradd\(8\)](#)
- [userdel\(8\)](#)
- [userinfo\(8\)](#)
- [usermod\(8\)](#)

10.8.2.1 - Actually adding users

Being that `user(8)` is not interactive, the easiest way to add users efficiently is to use the `adduser(8)` command. The actual command `/usr/sbin/user` is just a frontend to the rest of the `/usr/sbin/user*` commands. Therefore, the following commands can be added by using **user add** or **useradd**, its your choice as to what you want, and doesn't change the use of the commands at all.

In this example, we are adding the same user with the same specifications as the user that was added [above](#). `useradd(8)` is much easier to use if you know the default setting before adding a user. These settings are located in `/etc/usermgmt.conf` and can be viewed by doing so:

```

$ user add -D
group          users
base_dir       /home
skel_dir       /etc/skel
shell          /bin/csh
inactive       0
expire         Null (unset)
range          1000..60000

```

The above settings are what will be set unless you specify different with command line options. For example, in our case, we want the user to go to the group **guest**, not **users**. One more little hurdle with adding users, is that passwords must be specified on the commandline. This is, the encrypted passwords, so you must first use the [encrypt\(1\)](#) utility to create the password. For example:

OpenBSD's passwords by default use the Blowfish algorithm for 6 rounds. Here is an example line to create an encrypted password to specify to `useradd(8)`.

```

$ encrypt -p -b 6
Enter string:
$2a$06$Y0dOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRz1q

```

Now that we have our encrypted password, we are ready to add the user.

```

# user add -p '$2a$06$Y0dOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRz1q' -u 1002
\  

-s /bin/ksh -c "Test FAQ User" -m -g guest testuser

```

Note: Make sure to use `"` around the password string, not `'` as the shell will interpret these before sending it to `user(8)`. In addition to that, make sure you specify the `-m` option if you want the user's home directory created and the files from `/etc/skel` copied over.

To see that the user was created correctly, we can use many different utilities. Below are a few commands you can use to quickly check that everything was created correctly.

```
$ ls -la /home
total 14
drwxr-xr-x  5 root      wheel   512 May 12 14:29 .
drwxr-xr-x 15 root      wheel   512 Apr 25 20:52 ..
drwxr-xr-x 24 ericj     wheel  2560 May 12 13:38 ericj
drwxr-xr-x  2 testuser  guest   512 May 12 14:28 testuser
$ id testuser
uid=1002(testuser) gid=31(guest) groups=31(guest)
$ finger testuser
Login: testuser                Name: Test FAQ User
Directory: /home/testuser      Shell: /bin/ksh
Last login Sat Apr 22 16:05 (EDT) on ttyC2
No Mail.
No Plan.
```

In addition to these commands, `user(8)` provides its own utility to show user characteristics, called `userinfo(8)`.

```
$ userinfo testuser
login    testuser
passwd   *
uid      1002
groups   guest
change   Wed Dec 31 19:00:00 1969
class
gecos    Test FAQ User
dir      /home/testuser
shell    /bin/ksh
expire   Wed Dec 31 19:00:00 1969
```

10.8.2.2 - Removing users

To remove users with the `user(8)` hierarchy of commands, you will use `userdel(8)`. This is a very simple, yet useable command. To remove the user created in the last example, simply:

```
# userdel -r testuser
```

Notice the `-r` option, which must be specified if you want the users home directory to be deleted as well. Alternatively, you can specify `-p` and not `-r` and this will lock the user's account, but not remove any information.

10.9 - How do I create an ftp-only account (not anonymous FTP!)?

There are a few ways to do this, but a very common way to do such is to add `/bin/false` into `/etc/shells`. Then when you create the user set his shell to `/bin/false`, they will not be able log in interactively, but will be able to use ftp capabilities. [adduser\(8\)](#) will give them a home dir by default of `/home/<user>`. If this is what you desire it doesn't need to be changed, however you can set this to whatever directory you wish. You can force this user to only be able to see files in their home directory by adding their username to `/etc/ftpchroot`. Using the `-A` option to [ftpd\(8\)](#), you can allow only ftpchroot logins!

10.10 - Setting up Quotas

Quotas are used to limit users space that they have available to them on your drives. It can be very helpful in situations where you have limited resources. Quotas can be set in two different ways.

- User Quotas
- Group Quotas

The first step to setting up quotas is to make sure that option `QUOTA` is in your Kernel Configuration. This option is in the `GENERIC` kernel. After this you need to mark in `/etc/fstab` the filesystems which will have quotas enabled. The keywords `userquota` and `groupquota` should be used to mark each fs that you will be using quotas on. By default the files `quota.user` and `quota.group`

will be created at the root of that filesystem to hold the quota information. This default can be overwritten by doing, `userquota=/var/quotas/quota.user`. Or whatever file you want to use for holding quota information. Here is an example `/etc/fstab` that has one filesystem with userquotas enabled.

```
/dev/wd0a / ffs rw,userquota=/var/quotas/quota.user 1 1
```

Now it's time to set the user's quotas. To do so you use the utility [edquota\(8\)](#). A simple use is just `edquota <user>`. `edquota(8)` will use `vi(1)` to edit the quotas unless the environmental variable `EDITOR` is set to a different editor. For example:

```
# edquota ericj
```

This will give you output similar to this:

```
Quotas for user ericj:
/: blocks in use: 62, limits (soft = 0, hard = 0)
   inodes in use: 25, limits (soft = 0, hard = 0)
```

To add limits, edit it to give results like this:

```
Quotas for user ericj:
/: blocks in use: 62, limits (soft = 1000, hard = 1050)
   inodes in use: 25, limits (soft = 0, hard = 0)
```

In this the softlimit is set to 1000 blocks and the hardlimit is set to 1050 blocks. A softlimit is a limit where the user is just warned when they cross it and have until their grace period is up to get their disk usage below their limit. Grace periods can be set by using the `-t` option on `edquota(8)`. After the grace period is over the softlimit is handled as a hardlimit. This usually results in an allocation failure.

Now that the quotas are set, you need to turn the quotas on. To do this use [quotaon\(8\)](#). For example:

```
# quotaon -a
```

This will go through `/etc/fstab` to turn on the filesystems with quota options. Now that quotas are up and running, you can view them by the [quota\(1\)](#). Using a command of `quota <user>` will give that users information. When called with no arguments the quota command will give your quota statistics. For example:

```
# quota ericj
```

Will result in output similar to this:

```
Disk quotas for user ericj (uid 1001):
  Filesystem  blocks  quota  limit  grace  files  quota  limit  grace
  /           62     1000  1050   0      27     0     0     0
```

By default quotas set in `/etc/fstab` will be started on boot. To turn them off use

```
# quotaoff -a
```

10.11 - How to Setup Kerberos Clients and Servers under OpenBSD

As a user/administrator of OpenBSD systems, you are fortunate that KerberosIV is an pre-installed component of the default system. Here is a guide to setting up both the Kerberos realm server, as well as a client.

An ***EXTREMELY*** important point to remember is that Kerberos clients and servers must have their system clocks synchronized. If there is more than a 5 minute time skew, you will receive wierd errors that do not immediately reveal themselves to be caused by time skew, such as:

```
kinit: Can't send request (send_to_kdc)
```

Another more accurate error is:

```
kauth: Time is out of bounds (krb_rd_req)
```

An easy way to synchronize system clocks is with `xntpd`, available in the ports tree at `/usr/ports/sysutils/xntpd/`.

This FAQ entry assumes you have prior knowledge of the Kerberos concepts. For a great, easy to understand, reference, see:

- [The FreeBSD handbook](#)
- Use the command `info kth-krb`
- [Designing an Authentication System: a Dialogue in Four Scenes](#)

- [Papers and Documentation Describing KerberosIV](#)

Or the book

- [Network Security Private Communication in a Public World \[Kaufman, Perlman, Speciner, 1995\]](#)

How to setup the Kerberos IV REALM and SERVER

We will be setting up the CIARASYSTEMS.COM realm, with `avalanche.ciarasystems.com` as the main server.

To start off, we will need to edit our configuration files. These files are located at `/etc/kerberosIV/`. The two files we are concerned about are `krb.realms` and `krb.conf`. Let's start off with `krb.conf`.

```
[root@avalanche kerberosIV] cat krb.conf
CIARASYSTEMS.COM
CIARASYSTEMS.COM avalanche.ciarasystems.com admin server
```

As you can see, this tells kerberos that the domain is CIARASYSTEMS.COM (or logical realm) and that within that domain, `avalanche` is the administration server. Next we will look at `krb.realms`. For more information on this refer to [krb.conf](#).

```
[root@avalanche kerberosIV] cat krb.realms
avalanche.ciarasystems.com      CIARASYSTEMS.COM
.ciarasystems.com              CIARASYSTEMS.COM
```

`krb.realms` provides a translation from a hostname to the Kerberos realm name for the services provided by that host. Each line of the translation file is in one of the following forms (domain_name should be of the form `.XXX.YYY`). So in this example, `avalanche` is the hostname of a computer on the CIARASYSTEMS.COM realm. And `.ciarasystems.com` is the domain name on the realm CIARASYSTEMS.COM. Again, for further information read the [krb.realms](#) man page.

Next we will run [kdb_init\(8\)](#) to create the initial Kerberos database.

```
[root@avalanche kerberosIV] kdb_init
Realm name [default NO.DEFAULT.REALM ]: CIARASYSTEMS.COM
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
```

```
Enter Kerberos master password: not shown
Verifying password -
Enter Kerberos master password:
```

Next we need to use [kstash\(8\)](#) which is used to save the Kerberos key distribution center (KDC) database master key in the master key cache file.

```
[root@avalanche kerberosIV] kstash
Enter Kerberos master password:
```

```
Current Kerberos master key version is 1.
```

```
Master key entered.  BEWARE!
Wrote master key to /etc/kerberosIV/master_key
This saves the encrypted master password in /etc/kerberosIV/master_key.
```

Next, we need two principals to be added to the database for each system that will be secured with Kerberos. Their names are `kpasswd` and `rcmd`. These two principals are made for each system, with the instance being the name of the individual system. These daemons, `kpasswd` and `rcmd` allow other systems to change Kerberos passwords and run commands like `rcp`, `rlogin` and `rsh`.

```
# kdb_edit
Opening database...
```

```
Enter Kerberos master key:
```

```
Current Kerberos master key version is 1.
```

```
Master key entered.  BEWARE!
```

10.0 - System Management

Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: **passwd**
Instance: **avalanche**

<Not found>, Create [y] ? **y**

Principal: passwd, Instance: avalanche, kdc_key_ver: 1
New Password: <----- Use 'RANDOM' as password
Verifying password -
New Password:

Random password [y] ? **y**

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [1999-12-31] ? **2001-12-31**
Max ticket lifetime (*5 minutes) [255] ?
Attributes [0] ?
Edit O.K.

Principal name: **rcmd**
Instance: **avalanche**

<Not found>, Create [y] ? **y**

Principal: rcmd, Instance: avalanche, kdc_key_ver: 1
New Password: <----- Use 'RANDOM' as password
Verifying password -
New Password:

Random password [y] ? **y**

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [1999-12-31] ? **2001-12-31**
Max ticket lifetime (*5 minutes) [255] ?
Attributes [0] ?
Edit O.K.
Principal name: <----- Hit <ENTER> to end

A srvtab file is the service key file. These must be extracted from the Kerberos key distribution center database in order for services to authenticate using Kerberos. For each hostname specified on the command line, [ext_srvtab\(8\)](#) creates the service key file `hostname-new-srvtab`, containing all the entries in the database with an instance field of hostname.

```
[root@avalanche kerberosIV] ext_srvtab avalanche
```

Enter Kerberos master password:

Current Kerberos master key version is 1.

Master key entered. BEWARE!
Generating 'avalanche-new-srvtab'....

```
[root@avalanche kerberosIV] mv avalanche-new-srvtab srvtab  
[root@avalanche kerberosIV] chmod 600 srvtab
```

Now we can add users to our database.

```
[root@avalanche kerberosIV] kdb_edit  
Opening database...
```

```
Enter Kerberos master key:
```

```
Current Kerberos master key version is 1.
```

```
Master key entered. BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.
```

```
Principal name: jeremie
Instance:
```

```
<Not found>, Create [y] ? y
```

```
Principal: jeremie, Instance: , kdc_key_ver: 1
New Password: <---- enter a secure password here
Verifying password
```

```
New Password: <---- re-enter the password here
```

```
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
```

```
Principal name: <---- null entry here will cause an exit
or you can add more entries.
```

So now all the Kerberos particulars are setup. All that is left is to enable boot-time loading of the Kerberos server and to enable Kerberized-daemons.

In /etc/rc.conf, set:

```
kerberos_server=YES
```

In /etc/inetd.conf, uncomment:

```
telnet      stream  tcp    nowait  root    /usr/libexec/telnetd    telnetd -k
klogin      stream  tcp    nowait  root    /usr/libexec/rlogind    rlogind -k
kshell      stream  tcp    nowait  root    /usr/libexec/rshd       rshd -k
kauth       stream  tcp    nowait  root    /usr/libexec/kauthd     kauthd
```

Then, either reboot, or:

```
[root@avalanche /] kill -HUP `cat /var/run/inetd.pid`
[root@avalanche /] /usr/libexec/kerberos >> /var/log/kerberos.log &
[root@avalanche /] /usr/libexec/kadmind -n >> /var/log/kadmind.log &
```

Note: this is a rather simple server setup. Usually, redundant servers are setup (as slave servers) so that if one server goes down, all the services that depend on Kerberos don't go down. We can also add 'su' privileges to a specific principal, see [the FreeBSD Handbook](#).

How to kerberize your client workstation

We will be setting the workstation named *gatekeeper* to be in the CIARASYSTEMS.COM realm, with avalanche.ciarasystems.com as the main server.

To start off, we need to setup our *krb.conf* and *krb.realms* like the above machine. This is so *gatekeeper* will know what server is the KDC and what domain it is on. Again here are the file contents.

```
[root@gatekeeper kerberosIV] cat krb.conf
CIARASYSTEMS.COM
CIARASYSTEMS.COM avalanche.ciarasystems.com admin server
```

```
[root@gatekeeper kerberosIV] cat krb.realms
avalanche.ciarasystems.com      CIARASYSTEMS.COM
.ciarasystems.com              CIARASYSTEMS.COM
```

Now that is set up, we need to initialize kerberos. To obtain a ticket you use [kinit\(1\)](#).

```
xyz:jeremie% kinit
The OpenBSD Project (gatekeeper)
Kerberos Initialization
Kerberos name: jeremie
Password:
```

Now we have identified we can list our tickets with [klist\(1\)](#).

```
xyz:jeremie$ klist
Ticket file:      /tmp/tkt1000
Principal:       jeremie@CIARASYSTEMS.COM

      Issued                Expires                Principal
Jun 28 01:03:25  Jun 28 11:03:25  krbtgt.CIARASYSTEMS.COM@CIARASYSTEMS.COM
```

Looks like we are set now. All that's left to do is test it. Here we will test it with [rlogin\(1\)](#) and [telnet\(1\)](#).

```
xyz:jeremie% telnet avalanche
Trying 192.168.0.38...
Connected to avalanche.
Escape character is '^]'.
[ Trying mutual KERBEROS4 ... ]
[ Kerberos V4 accepts you ]
[ Kerberos V4 challenge successful ]
Last login: Sun Jun 27 22:52:25 on ttyt1 from gatekeeper
Warning: no Kerberos tickets issued.
OpenBSD 2.5 (AVALANCHE) #5: Tue Apr  6 01:18:16 EDT 1999
```

and

```
xyz:jeremie% rlogin avalanche
Last login: Sun Jun 27 22:53:39 on ttyt1 from gatekeeper
Warning: no Kerberos tickets issued.
OpenBSD 2.5 (AVALANCHE) #5: Tue Apr  6 01:18:16 EDT 1999
```

We can tell that it is indeed using Kerberos to authenticate the rlogin session. To get rid of any tickets issued, you would use [kdestroy\(1\)](#). For example:

```
xyz:jeremie% kdestroy
Tickets destroyed.
xyz:jeremie% rlogin avalanche
krccmd: No ticket file (tf_util)
rlogin: warning, using standard rlogin: can't provide Kerberos auth data.
avalanche: Connection refused
```

Do not worry about 'Warning: no Kerberos tickets issued.' This is because we're only doing kerberos authentication, not ticket passing. If you want ticket passing, use [OpenSSH](#) which has support. Stock KerberosIV doesn't have support for tgt passing, either - only the AFS kaserver's implementation of krb4, since the regular KerberosIV kdc checks client IP address listed in the ticket.

10.12 - Setting up Anonymous FTP Services.

Anonymous FTP allows users without accounts to access files on your computer via the File Transfer Protocol. This will give an overview of setting up the anonymous FTP server, and its logging, etc.

Adding the FTP account

To start off, you need to have an account on your system of "ftp". This account shouldn't have a useable password. Here we will set the login directory to /home/ftp, but you can put it wherever you want. When using anonymous ftp, the ftp daemon will chroot itself to the home directory of the 'ftp' user. To read up more on that, read the [ftp\(8\)](#) and [chroot\(2\)](#) man pages. Here is an example of adding the ftp user. I will do this using [adduser\(8\)](#). We also need to add /bin/false to our */etc/shells*, this is the "shell" that we will be giving to the ftp user. This won't allow them to login, even though we will give them an empty password. To do this you can simply *echo /bin/false >>*

`/etc/shells`. Also if you wish for that shell to show up during the adduser questions, you need to modify `/etc/adduser.conf`.

```
oshibana# adduser
```

```
Use option ``-silent'' if you don't want see all warnings & questions.
```

```
Reading /etc/shells
```

```
Check /etc/master.passwd
```

```
Check /etc/group
```

```
Ok, let's go.
```

```
Don't worry about mistakes. I will give you the chance later to correct any input.
```

```
Enter username [a-z0-9_]: ftp
```

```
Enter full name []: anonymous ftp
```

```
Enter shell csh false ksh nologin sh tcsh zsh [sh]: false
```

```
Uid [1002]:
```

```
Login group ftp [ftp]:
```

```
Login group is ``ftp''. Invite ftp into other groups: guest no  
[no]: no
```

```
Enter password []:
```

```
Use an empty password? (y/n) [y]: y
```

```
Name: ftp
```

```
Password: ****
```

```
Fullname: anonymous ftp
```

```
Uid: 1002
```

```
Gid: 1002 (ftp)
```

```
Groups: ftp
```

```
HOME: /home/ftp
```

```
Shell: /usr/bin/false
```

```
OK? (y/n) [y]: y
```

```
Added user ``ftp''
```

```
Copy files from /usr/share/skel to /home/ftp
```

```
Add another user? (y/n) [y]: n
```

```
Goodbye!
```

Directory Setup

Along with the user, this created the directory `/home/ftp`. This is what we want, but there are some changes that we will have to make to get it ready for anonymous ftp. Again these changes are explained in the [ftp\(8\)](#) man page.

You **do not** need to make a `/home/ftp/usr` or `/home/ftp/bin` directory.

- `/home/ftp` - This is the main directory. It should be owned by root and have permissions of 555.
- `/home/ftp/etc` - This is entirely optional and not recommended, as it only serves to give out information on users which exist on your box. If you want your anonymous ftp directory to appear to have real users attached to your files, you should copy `/etc/pwd.db` and `/etc/group` to this directory. This directory should be mode 511, and the two files should be mode 444. These are used to give owner names as opposed to numbers. There are no passwords stored in `pwd.db`, they are all in `spwd.db`, so don't copy that over.
- `/home/ftp/pub` - This is a standard directory to place files in which you wish to share. This directory should also be mode 555.

Note that all these directories should be owned by "root". Here is a listing of what the directories should look like after their creation.

```
oshibana# pwd
```

```
/home
```

```
oshibana# ls -laR ftp
```

```
total 5
```

```
dr-xr-xr-x  5 root  ftp    512 Jul  6 11:33 .
```

```
drwxr-xr-x  7 root  wheel  512 Jul  6 10:58 ..
```

```
dr-x--x--x  2 root  ftp    512 Jul  6 11:34 etc
```

```
dr-xr-xr-x  2 root  ftp    512 Jul  6 11:33 pub
```

```
ftp/etc:
```

```
total 43
dr-x--x--x  2 root  ftp    512 Jul  6 11:34 .
dr-xr-xr-x  5 root  ftp    512 Jul  6 11:33 ..
-r--r--r--  1 root  ftp    316 Jul  6 11:34 group
-r--r--r--  1 root  ftp  40960 Jul  6 11:34 pwd.db
```

```
ftp/pub:
total 2
dr-xr-xr-x  2 root  ftp    512 Jul  6 11:33 .
dr-xr-xr-x  5 root  ftp    512 Jul  6 11:33 ..
```

Starting up the server and logging

With `ftpd` you can choose to either run it from `inetd` or the `rc` scripts can kick it off. These examples will show our daemon being started from [inetd.conf](#). First we must become familiar with some of the options to `ftpd`. The default line from `/etc/inetd.conf` is:

```
ftp          stream  tcp     nowait  root    /usr/libexec/ftpd      ftpd -US
```

Here `ftpd` is invoked with `-US`. This will log anonymous connections to `/var/log/ftpd` and concurrent sessions to `/var/run/utmp`. That will allow for these sessions to be seen via `who(1)`. For some, you might want to run only an anonymous server, and disallow `ftp` for users. To do so you should invoke `ftpd` with the `-A` option. Here is a line that starts `ftpd` up for anonymous connections only. It also uses `-ll` which logs each connection to `syslog`, along with the `get`, `retrieve`, etc, `ftp` commands.

```
ftp          stream  tcp     nowait  root    /usr/libexec/tcpd      ftpd -llUSA
```

Note - For people using HIGH traffic `ftp` servers, you might want to not invoke `ftpd` from `inetd.conf`. The best option is to comment the `ftpd` line from `inetd.conf` and start `ftpd` from `rc.conf` along with the `-D` option. This will start `ftpd` as a daemon, and has much less overhead as starting it from `inetd`. Here is an example line to start it from `rc.conf`.

```
ftpd_flags="-DllUSA"          # for non-inetd use: ftpd_flags="-D"
```

This of course only works if you have `ftpd` taken out of `/etc/inetd.conf`.

Other relevant files

- `/etc/ftpwelcome` - This holds the Welcome message for people once they have connected to your `ftp` server.
- `/etc/motd` - This holds the message for people once they have successfully logged into your `ftp` server.
- `.message` - This file can be placed in any directory. It will be shown once a user enters that directory.

10.13 - Confining users to their home dir's in ftpd(8).

OpenBSD's [ftpd\(8\)](#) is setup by default to be able to handle this very easily. This is accomplished via the file `/etc/ftpchroot`. Since users cannot always be trusted, it might be necessary to restrain them to their home directories. This behavior is NOT on by default. Here is an example of what the default behavior is like.

```
$ ftp localhost
Connected to localhost.
220 oshibana FTP server (Version 6.4/OpenBSD) ready.
Name (localhost:ericj): ericj
331 Password required for ericj.
Password: *****
230- OpenBSD 2.6 (GENERIC) #690: Fri Oct 29 16:32:17 MDT 1999
230-
230- Welcome to OpenBSD: The proactively secure Unix-like operating system.
230-
230- Please use the sendbug(1) utility to report bugs in the system.
230- Before reporting a bug, please try to reproduce it with the latest
230- version of the code. With bug reports, please try to ensure that
230- enough information to reproduce the problem is enclosed, and if a
230- known fix for it exists, include that as well.
230-
230 User ericj logged in.
```

10.0 - System Management

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /
250 CWD command successful.
ftp> ls
227 Entering Passive Mode (127,0,0,1,60,7)
150 Opening ASCII mode data connection for 'file list'.
altroot
bin
dev
etc
home
mnt
root
sbin
stand
tmp
usr
var
bsd
sys
boot
226 Transfer complete.
ftp> quit
221 Goodbye.
```

As you can see here, access is granted to the whole server. In a perfect world this is ok, where all users can be trusted, but this isn't so. To limit a user, simply add their name to the file `/etc/ftpchroot`. Here is an example showing user "ericj" being restricted.

```
$ cat /etc/ftpchroot
#      $OpenBSD: faq10.html,v 1.45 2000/06/14 00:56:26 miod Exp $
#
# list of users (one per line) given ftp access to a chrooted area.
# read by ftpd(8).
ericj
```

This is enough to keep the user "ericj" from escaping from his own directory. As you can see in the next example. The `/` directory has suddenly changed to his home dir!

```
$ ftp localhost
Connected to localhost.
220 oshibana FTP server (Version 6.4/OpenBSD) ready.
Name (localhost:ericj): ericj
331 Password required for ericj.
Password: *****
230 User ericj logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /
250 CWD command successful.
ftp> ls
227 Entering Passive Mode (127,0,0,1,92,171)
150 Opening ASCII mode data connection for 'file list'.
.login
.mailrc
.profile
.rhosts
.ssh
.cshrc
work
mail
```

10.0 - System Management

```
src  
226 Transfer complete.  
ftp> quit  
221 Goodbye.
```

[\[Back to Main Index\]](#) [\[To Section 9.0 - Migrating from Linux\]](#) [\[To Section 11.0 - Performance Tuning\]](#)



[www@openbsd.org](http://www.openbsd.org)

\$OpenBSD: faq10.html,v 1.45 2000/06/14 00:56:26 miod Exp \$

12.0 - For Advanced Users

Table of Contents

- [12.1 - Forcing DMA access for IDE disks](#)
 - [12.2 - Upgrading from various versions of OpenBSD via CVS.](#)
-

By *advanced users*, we mean to imply people who have a working knowledge of how to use and administrate a Unix-like operating system. If you follow instructions in this section without understanding what you are doing, you may cause problems in the operation of your system.

12.1 - Forcing DMA access for IDE disks

With the PCI IDE code, your chipset may not be known. If so, you will get a message like:

```
pciide0: DMA, (unused)
```

If you get this message, you can try and force DMA mode by using 'flags 0x0001' on your pciide entry in your kernel config file. That would look something like this:

```
pciide* at pci ? dev ? function ? flags 0x0001
```

After doing this, the pciide code will try to use DMA mode regardless of whether or not it actually knows how to do so with your chipset. If this works, and your system makes it through fsck and startup, it is likely that this will work for good. If this does not work, and the system hangs or panics after booting, then you can't use DMA mode (yet, until support is added for your chipset). Note that if you find the documentation for your PCI-IDE controller's chipset, this is a good start to fully supporting your chipset within the PCI-IDE code. You can look on the manufacturer's website or call them. If your PCI-IDE controller is part of your motherboard, figure out who manufactures the chipset and pursue their resources!

Note that you will know that DMA support has been enabled if you see this message:

```
cd0(pciide0:1:0): using PIO mode 3, DMA mode 1 (using DMA data transfers)
```

This means that pciide0, channel 1, drive 0 (which appears to be an ATAPI CD-ROM) is using DMA data transfers.

Some notes:

DMA is not supported on wdc* unless a DMA channel (drq) is specified. I'm not sure what the "standard" drqs are for hard disk controllers. To enable

```
wdc0 at isa? port 0x1f0 irq 14 flags 0x00
```

Non-ultra DMA does not necessarily lead to higher bandwidth vs PIO. However, it decreases the CPU load significantly.

[\[Back to Main Index\]](#) [\[To Section 11.0 - Performance Tuning\]](#) [\[To Section 13.0 - IPsec\]](#)



[\[Back to Main Index\]](#) [\[To Section 12.0 - For Advanced Users\]](#) [\[To Section 14.0 - Using disks in OpenBSD\]](#)

Portions of this document were taken from:

- [ipsec\(4\)](#)
- [vpn\(8\)](#)
- [IPSec for Dummies](#) by Julian Elischer
- [ISAKMP Howto](#) by Patrick Ethier
- [X.509v3 certificates with isakmpd](#) by Jörgen Granstam

13.0 - Using IPsec (IP Security Protocol)

- [13.1 - What is IPsec?](#)
- [13.2 - That's nice, but why do I want to use IPsec?](#)
- [13.3 - What are the protocols behind IPsec?](#)
- [13.4 - On the wire format](#)
- [13.5 - Configuring IPsec](#)
- [13.6 - How do I setup IPsec with manual keying?](#)
- [13.7 - How do I setup photurisd?](#)
- [13.8 - How do I setup isakmpd?](#)
- [13.9 - How do I use isakmpd with X.509 certificates?](#)
- [13.10 - What IKE clients are compatible with isakmpd?](#)
- [13.11 - Troubleshooting IPsec/VPN](#)
- [13.12 - Related Documentation](#)

13.1 - What is IPsec?

IPsec is a set of extensions to the IP protocol family. It provides cryptographic security services. These services allow for authentication, integrity, access control, and confidentiality. IPsec provides similar services as SSL, but at the network layer, in a way that is completely transparent to your applications, and much more powerful. We say this because your applications do not have to have any knowledge of IPsec to be able to use it. You can use [any IP protocol](#) over IPsec. You can create encrypted tunnels (VPNs), or just do encryption between computers. Since you have so many options, IPsec is rather complex (much more so than SSL!)

Before you start using IPsec, we strongly recommend that you check out the "recommended reading" of [part 6](#) of the FAQ. In particular, if you don't already understand it, the [Understanding IP Addressing](#) document is highly recommended.

In a logical sense, IPsec works in any of these three ways:

- Host-to-Host
- Host-to-Network
- Network-to-Network

In every scenario that involves a network, we mean to imply router. As in, Host-to-Router (and this router controls and encrypts traffic for a particular *Network*.)

As you can see, IPsec can be used to tunnel traffic for VPN connections. However, its utility reaches beyond VPNs. With a central Internet Key Exchange registry, every machine on the internet could talk to another one and employ powerful encryption



14.0 - Disk setup

Table of Contents

- [14.1 - Using OpenBSD's Disklabel](#)
 - [14.2 - Using OpenBSD's fdisk](#)
 - [14.3 - Adding extra disks in OpenBSD](#)
 - [14.4 - How to swap to a file](#)
 - [14.5 - Soft-updates](#)
 - [14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using partition 3 id 0".](#)
 - [14.7 - How do I get a dmesg from a boot floppy?](#)
 - [14.8 - Installing Bootblocks - i386 specific](#)
 - [14.9 - Preparing for disaster: Backing up and Restoring from tape.](#)
-

Using OpenBSD's Disklabel

Table of Contents

- [What is disklabel\(8\)?](#)
- [Disklabel during the OpenBSD install](#)
- [Common disklabel\(8\) uses.](#)

What is disklabel(8)?

Disklabel's are created to allow an efficient interface between your disk and the disk drivers contained within the kernel. Labels hold certain information about your disk, like your drive geometry and information about your filesystems. This is then used by the bootstrap program to load the drive and to know where filesystems are contained on the drive. Labels are also used in conjunction with the filesystems to create a more efficient environment. You can read more in-depth information about disklabel by reading the [disklabel\(5\)](#) man page.

As an additional gain, using disklabel helps overcome architecture limitations on disk partitioning. For example, on i386, you can only have 4 primary partitions. (Partitions that other operating systems, such as Windows NT or DOS can see.) With disklabel(8), you use one of these 'primary' partitions to store **all** of your OpenBSD partitions (eg. 'swap', '/', '/usr' and '/var'). And you still have 3 more partitions available for other OSs!

disklabel(8) during OpenBSD's install

One of the major parts of OpenBSD's install is your initial creation of labels. This comes (for i386 users) directly after using [fdisk\(1\)](#). During the install you use disklabel to create your separate labels which will contain your separate mountpoints. During the install, you can set your mountpoints from within disklabel(8), but this isn't completely necessary considering you will be prompted later to confirm you choices. But it does make your install go just a little smoother.

Since this is during the install you won't have any existing labels, and they will need to be created. The first label you will create is the label 'a'. This label SHOULD be your where / will be mounted. You can see recommended partitions that

should be created and their sizes by reading [faq4.3](#). For servers it is recommended that you create at least these label's separately. For desktop users creating one mountpoint at / will probably suffice. When initially creating your root partition ('a' label), keep in mind that you will need SOME space left for your swap label. Now that the basics have been explained, here is an example of using disklabel during an install. In this first example it is assumed that OpenBSD will be the only operating system on this computer, and that a full install will be done.

If this disk is shared with other operating systems, those operating systems should have a BIOS partition entry that spans the space they occupy completely. For safety, also make sure all OpenBSD file systems are within the offset and size specified in the 'A6' BIOS partition table. (By default, the disklabel editor will try to enforce this). If you are unsure of how to use multiple partitions properly (ie. seperating /, /usr, /tmp, /var, /usr/local, and other things) just split the space into a root and swap partition for now.

```
# using MBR partition 3: type A6 off 63 (0x3f) size 4991553 (0x4c2a41)
```

Treating sectors 63-16386300 as the OpenBSD portion of the disk.
You can use the 'b' command to change this.

Initial label editor (enter '?' for help at any prompt)

```
> d a
> a a
offset: [63] <Enter>
size: [16386237] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /
fragment size: [1024] <Enter>
block size: [8192] <Enter>
cpg: [16] <Enter>
> a b
offset: [131103] <Enter>
size: [16255197] 64M
Rounding to nearest cylinder: 131040
FS type: [swap] <Enter>
```

At this point we have created a 64M root partition mounted at /, and a 64Meg swap partition. Notice that the offset starts at sector 63. This is what you want. When it comes to the size, disklabel will show your size in sectors, however, you don't need to enter sizes in the same format. Like the example above you can enter sizes in the manner of *64 Megabytes = 64M* and *2 Gigabytes = 2G*. Disklabel will then round to the nearest cylinder. In the example above you will also notice that disklabel assumes that label 'b' will be a swap. This is a correct assumption as the GENERIC kernel is set to look for swap on label 'b', and you should just follow this guideline and use 'b' as your swap area.

The next example will take you through the creation of two more labels. This means that it's not a complete install, as the size of these won't be enough to install OpenBSD to its fullest. Showing the creation of all the partitions would just be repetitive.

```
> a d
offset: [262143] <Enter>
size: [16124157] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /tmp
fragment size: [1024] <Enter>
block size: [8192] <Enter>
```

```

cpg: [16] <Enter>
> a e
offset: [393183] <Enter>
size: [15993117] 64M
Rounding to nearest cylinder: 131040
FS type: [4.2BSD] <Enter>
mount point: [none] /var
fragment size: [1024] <Enter>
block size: [8192] <Enter>
cpg: [16] <Enter>

```

In the above example, there are two things you might notice. One being that the offset is automatically figured out for you to be the next in order. When doing an install of this sort, you won't need to mess with changing the offsets at all. Another difference you might notice will be that label 'c' has been skipped. This is done for a reason, which is that label 'c' is a label that represents the whole disk. For this reason you shouldn't deal with label 'c' in any way.

Once all your labels have been created all that's left to do is write the labels to disk, and move on in the installation process. To write everything and quit disklabel (and continue with the install) do:

```

> w
> q

```

NOTE - For user's with large drives. If your bios isn't able to support a drive of that size OpenBSD cannot support it either. Otherwise OpenBSD should be able to handle your drive just fine. If you are in a situation where your bios doesn't support your drive, you can try Maxtor EZ-Drive or other similar overlay product.

Common uses for disklabel(8)

Once your system is installed, you shouldn't need to use disklabel too often. But some times you will need to use disklabel are when adding, removing or restructuring your disks. One of the first things you will need to do is view your current disklabel. To do this simply type:

```

# disklabel wd0 <----- Or whatever disk device you'd like to view

# using MBR partition 3: type A6 off 64 (0x40) size 16777152 (0xffffc0)
# /dev/rwd0c:
type: ESDI
disk:
label: TOSHIBA MK2720FC
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 2633
total sectors: 2654064
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0          # milliseconds
track-to-track seek: 0 # milliseconds
drivedata: 0

16 partitions:
#          size  offset  fstype  [fsize bsize  cpg]

```

```

a:  2071440      65583      4.2BSD      1024  8192      16   # (Cyl.   65*- 2120)
b:    65520       63         swap                # (Cyl.    0*- 65)
c:  2654064       0        unused             0     0         # (Cyl.    0 - 2632)
j:   512001  2137023      4.2BSD      1024  8192      16   # (Cyl. 2120*- 2627*)

```

The above command simply allows you to view the existing disklabel, and assuring that you dont mess anything up. (Which we all need sometimes.) But to be able to make changes you must use the -E option with disklabel like so:

```
# disklabel -E wd0
```

This will bring you to a prompt, the same as the one that you used during the OpenBSD install. Probably the single most important command at this prompt is '?'. This will give you a list of possible options pertaining to disklabel. You can even view the entire disklabel(8) man page with the 'M' command. From this prompt, you will do all of your adding, deleting and changing of partitions. For additional information read the [disklabel\(8\)](#) man page.

14.2 - Using fdisk

First be sure to check the fdisk main page. [fdisk\(8\)](#)

Fdisk is a program to help with the maintenance of your partitions. This program is used at install time to set up your OpenBSD partition (this partition can contain several labels, each with filesystems/swap/etc.). It can divide space on your drives and set one active. This program will usually be used in Single User Mode (boot -s). Fdisk also sets the MBR on your various hard disks.

For installation purposes, most times you'll only need **ONE** OpenBSD partition, and then using disklabel to put a swap and a filesystem on it.

To just view your partition table using fdisk just use:

```
# fdisk fd0
```

Which will give an output similar to this:

```

Disk: fd0          geometry: 80/2/18 [2880 sectors]
Offset: 0         Signatures: 0xAA55,0x0

      Starting      Ending
  #:  id  cyl  hd sec -  cyl  hd sec [      start -      size]
-----
*0:  A6   0   0   1 -   79   1  18 [      0 -      2880] OpenBSD
 1:   00   0   0   0 -    0   0   0 [      0 -          0] unused
 2:  A7   0   0   2 -   79   1  18 [      1 -      2879] NEXTSTEP
 3:   00   0   0   0 -    0   0   0 [      0 -          0] unused

```

In this example we are viewing the fdisk output of floppy drive. We can see the OpenBSD partition (A6) and its size. The * tells us that the OpenBSD partition is a bootable partition.

In the previous example we just viewed our information. What if we want to edit our partition table? Well, to do so we must use the -e flag. This will bring up a command line prompt to interact with fdisk.

```
# fdisk -e wd0
```

```
Enter 'help' for information
```

```
fdisk: 1> help
```

```

help          Command help list
manual        Show entire OpenBSD man page for fdisk
reinit        Re-initialize loaded MBR (to defaults)
disk          Edit current drive stats
edit          Edit given table entry
flag          Flag given table entry as bootable

```

```

update      Update machine code in loaded MBR
select     Select extended partition table entry MBR
print      Print loaded MBR partition table
write      Write loaded MBR to disk
exit       Exit edit of current MBR, without saving changes
quit       Quit edit of current MBR, saving current changes
abort      Abort program without saving current changes

```

```
fdisk: 1>
```

It is perfectly safe in fdisk to go in and explore, just make sure to answer **N** to saving the changes and ***DON'T*** use the **write** command.

Here is an overview of the commands you can use when you choose the **-e** flag.

- **help** Display a list of commands that fdisk understands in the interactive edit mode.
- **reinit** Initialize the currently selected, in-memory copy of the boot block.
- **disk** Display the current drive geometry that fdisk has probed. You are given a chance to edit it if you wish.
- **edit** Edit a given table entry in the memory copy of the current boot block. You may edit either in BIOS geometry mode, or in sector offsets and sizes.
- **flag** Make the given partition table entry bootable. Only one entry can be marked bootable. If you wish to boot from an extended partition, you will need to mark the partition table entry for the extended partition as bootable.
- **update** Update the machine code in the memory copy of the currently selected boot block.
- **select** Select and load into memory the boot block pointed to by the extended partition table entry in the current boot block.
- **print** Print the currently selected in-memory copy of the boot block and its MBR table to the terminal.
- **write** Write the in-memory copy of the boot block to disk. You will be asked to confirm this operation.
- **exit** Exit the current level of fdisk, either returning to the previously selected in-memory copy of a boot block, or exiting the program if there is none.
- **quit** Exit the current level of fdisk, either returning to the previously selected in-memory copy of a boot block, or exiting the program if there is none. Unlike exit it does write the modified block out.
- **abort** Quit program without saving current changes.

14.3 - Adding extra disks in OpenBSD

Well once you get your disk installed **PROPERLY** you need to use [fdisk\(8\)](#) (*i386 only*) and [disklabel\(8\)](#) to set up your disk in OpenBSD.

For i386 folks, start with fdisk. Other architectures can ignore this.

```
# fdisk -i sd2
```

This will initialize the disk's "real" partition table for exclusive use by OpenBSD. Next you need to create a disklabel for it. This will seem confusing.

```
# disklabel -e sd2
```

```
(screen goes blank, your $EDITOR comes up)
```

```
type: SCSI
```

```
...bla...
```

```
sectors/track: 63
```

```
total sectors: 6185088
```

```
...bla...
```

```
16 partitions:
```

#	size	offset	fstype	[fsize	bsize	cpg]	
c:	6185088	0	unused	0	0		# (Cyl. 0 - 6135)
d:	1405080	63	4.2BSD	1024	8192	16	# (Cyl. 0*- 1393*)
e:	4779945	1405143	4.2BSD	1024	8192	16	# (Cyl. 1393*- 6135)

First, ignore the 'c' partition, it's always there and is for programs like `disklabel` to function! For normal operations, `fsize` should always be 1024, `bsize` should always be 8192, and `cpg` should always be 16. `Fstype` is 4.2BSD. Total sectors is the total size of the disk. Say this is a 3 gigabyte disk. Three gigabytes in disk manufacturer terms is 3000 megabytes. So divide 6185088/3000 (use `bc(1)`). You get 2061. So to make up partition sizes for a, d, e, f, g, ... just multiply $X*2061$ to get X megabytes of space on that partition. The offset for your first new partition should be the same as the "sectors/track" reported earlier in `disklabel`'s output. For us it is 63. The offset for each partition afterwards should be a combination of the size of each partition and the offset of each partition (Except the C partition, since it has no play into this equation.)

Or, if you just want one partition on the disk, say you will use the whole thing for web storage or a home directory or something, just take the total size of the disk and subtract the sectors per track from it. $6185088-63 = 6185025$. Your partition is

```
d: 6185025      63      4.2BSD      1024  8192      16
```

If all this seems needlessly complex, you can just use `disklabel -E` to get the same partitioning mode that you got on your install disk! There, you can just use "96M" to specify "96 megabytes". (Or, if you have a disk big enough, 96G for 96 gigs!) Unfortunately, the `-E` mode uses the BIOS disk geometry, not the real disk geometry, and often times the two are not the same. To get around this limitation, type 'g d' for 'geometry disk'. (Other options are 'g b' for 'geometry bios' and 'g u' for geometry user, or simply, what the label said before `disklabel` made any changes.)

That was a lot. But you are not finished. Finally, you need to create the filesystem on that disk using [newfs\(8\)](#).

```
bsd# newfs wd1a
```

Or whatever your disk was named as per OpenBSD's disk numbering scheme. (Look at the output from `dmesg(1)` to see what your disk was named by OpenBSD.)

Now figure out where you are going to mount this new partition you just created. Say you want to put it on /u. First, make the directory /u. Then, mount it.

```
mount /dev/wd1a /u
```

Finally, add it to `/etc/fstab`

```
/dev/wd1a /u ffs rw 1 1
```

What if you need to migrate an existing directory like `/usr/local`? You should mount the new drive in `/mnt` and use `cpio -pdm` to copy `/usr/local` to the `/mnt` directory. Edit the `/etc/fstab` file to show that the `/usr/local` partition is now `/dev/wd1a` (your freshly formatted partition.) Example:

```
/dev/wd1a /usr/local ffs rw 1 1
```

Reboot into single user mode..**boot -s** Move the existing `/usr/local` to `/usr/local-backup` (or delete it if you feel lucky) and create an empty directory `/usr/local`. Then reboot the system, and whala!! the files are there!

14.4 - How to swap to a file

(Note: if you are looking to swap to a file because you are getting "virtual memory exhausted" errors, you should try raising the per-process limits first with `csh`'s [unlimit\(1\)](#), or `sh`'s [ulimit\(1\)](#).)

After the release of OpenBSD 2.5 came [swapctl\(8\)](#), which made dealing with swap devices much easier. If you are on an OpenBSD 2.5 system, exchange `swapctl` with `swapon`, and use `pstat -s` to list swap devices. Swapping to a file doesn't require a custom built kernel, although that can still be done, this faq will show you how to add swap space both ways.

Swapping to a file.

Swapping to a file is easiest and quickest way to get extra swap area's setup. This is not for users who are currently using Softupdates. (Which isn't enabled by default). To start out, you can see how much swap you are currently have and how much you are using with the [swapctl\(8\)](#) utility. You can do this by using the command:

```
ericj@oshibana> swapctl -l
Device          512-blocks      Used      Avail Capacity  Priority
swap_device     65520             8        65512    0%      0
```

This shows the devices currently being used for swapping and their current statistics. In the above example there is only one device named "swap_device". This is the predefined area on disk that is used for swapping. (Shows up as partition b when viewing disklabels) As you can also see in the above example, that device isn't getting much use at the moment. But for the purposes of this document, we will act as if an extra 32M is needed.

The first step to setting up a file as a swap device is to create the file. It's best to do this with the [dd\(1\)](#) utility. Here is an example of creating the file `/var/swap` that is 32M large.

```
ericj@oshibana> sudo dd if=/dev/zero of=/var/swap bs=1k count=32768
32768+0 records in
32768+0 records out
33554432 bytes transferred in 20 secs (1677721 bytes/sec)
```

Once this has been done, we can turn on swapping to that device. Use the following command to turn on swapping to this device

```
ericj@oshibana> sudo chmod 600 /var/swap
ericj@oshibana> sudo swapctl -a /var/swap
```

Now we need to check to see if it has been correctly added to the list of our swap devices.

```
ericj@oshibana> swapctl -l
Device          512-blocks      Used      Avail Capacity  Priority
swap_device     65520             8        65512    0%      0
/var/swap       65536             0        65536    0%      0
Total           131056            8        131048    0%
```

Now that the file is setup and swapping is being done, you need to add a line to your `/etc/fstab` file so that this file is configured on the next boot time also. If this line is not added, your won't have this swap device configured.

```
ericj@oshibana> cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/var/swap /var/swap sw 0 0
```

Swapping via a vnode device

This is a more permanent solution to adding more swap space. To swap to a file permanently, first make a kernel with `vnd0c` as swap. If you have `wd0a` as root filesystem, `wd0b` is the previous swap, use this line in the kernel configuration file (refer to compiling a new kernel if in doubt):

```
config          bsd          root on wd0a swap on wd0b and vnd0c dumps on wd0b
```

After this is done, the file which will be used for swapping needs to be created. You should do this by using the same command as in the above examples.

```
ericj@oshibana> sudo dd if=/dev/zero of=/var/swap bs=1k count=32768
32768+0 records in
32768+0 records out
33554432 bytes transferred in 20 secs (1677721 bytes/sec)
```

Now your file is in place, you need to add the file to you `/etc/fstab`. Here is a sample line to boot with this device as swap on

boot.

```
ericj@oshibana> cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/dev/vnd0c none swap sw 0 0
```

At this point your computer needs to be rebooted so that the kernel changes can take place. Once this has been done it's time to configure the device as swap. To do this you will use [vnconfig\(1\)](#).

```
ericj@oshibana> sudo vnconfig -c -v vnd0 /var/swap
vnd0: 33554432 bytes on /var/swap
```

Now for the last step, turning on swapping to that device. We will do this just like in the above examples, using [swapctl\(8\)](#). Then we will check to see if it was correctly added to our list of swap devices.

```
ericj@oshibana> sudo swapctl -a /dev/vnd0c
ericj@oshibana> swapctl -l
Device          512-blocks      Used      Avail Capacity  Priority
swap_device     65520            8         65512    0%        0
/dev/vnd0c      65536            0         65536    0%        0
Total           131056           8         131048    0%
```

14.5 - Soft-updates

Over the last few years Kirk McKusick has been working on something called "soft updates". This is based on an idea proposed by Greg Ganger and Yale Patt that imposing a partial ordering on the buffer cache operations would permit the requirement for synchronous writing of directory entries to be removed from the FFS code. Thus, a large performance increase of disk writing performance.

To enable Softupdates your kernel must have option

option FFS_SOFTUPDATES

then you need to boot into single-user mode:

```
boot>boot -s
[snip]
bsd# tunefs -s enable <raw device>
bsd# reboot -n
```

14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using partition 3 id 0".

This means that your MBR (Master Boot Record) was not installed properly or that your BIOS' idea of the hard disk geometry is not compatible with your current MBR. To solve this, first try to reinstall the OpenBSD boot blocks. To do so use [fdisk\(8\)](#) and [installboot\(8\)](#).

First, to get up and running, you will need to use your boot disk as a bootstrap. After you put the install disk in, and before it loads the kernel and ramdisk, you get a boot> prompt from the floppy disk. Use it to boot OpenBSD off of your hard disk.

```
booting...
OpenBSD boot 1.2.3
probing hd0 fd0...
boot> boot hd0a:/bsd
```

Now that you are booted, and assuming you have dedicated your entire hard disk to OpenBSD, re-initialize the Master Boot Record with [fdisk\(8\)](#). (If you have partitions on your hard disk for operating systems other than OpenBSD, you cannot use [installboot](#) anyways, you must look at another option such as OS-BS, described below.)

```
# fdisk -i wd0
```

Now, you need to write the boot blocks again.

```
# cp /usr/mdec/boot /boot
# /usr/mdec/installboot -v /boot /usr/mdec/biosboot wd0
```

Finally, reboot and test it out.

If this doesn't work you still have a few more options. So, luck hasn't run out yet. The first is to use a bootloader such as OS-BS. The OpenBSD cd has the `os-bs` bootloader included in the `tools` directory. If you didn't purchase a CD-ROM, you can download `os-bs` from any of the OpenBSD ftp sites. The file to grab is `pub/OpenBSD/2.6/tools/osbs135.exe`

Also, take some time to look through the OS-BS web pages at <http://www.prz.tu-berlin.de/~wolf/os-bs.html>

There are also other commercial bootloaders or lilo that you can use for multi-booting.

Here is a simple outline of how to get lilo onto your system.

- Boot a DOS floppy and do a `fdisk /MBR`. Make sure you do that on the drive you'll be booting from.
- Boot from a linux disk and install LILO & chain it to your OpenBSD boot block.

For further instructions read [INSTALL.linux](#)

14.7 - How do I get a dmesg from a boot floppy?

RAMDISK images (boot floppies) do not ship with the `dmesg` utility. They do, however, have the `/kern` filesystem mounted. To copy the `dmesg` information to a file, do a:

```
# cat /kern/msgbuf >mydmesg
```

Boot disks also have 'more' to page through the output:

```
# more /kern/msgbuf
```

Also check with [section 4.5](#)

14.8 - Installing Bootblocks - i386 specific

Older versions of MS-DOS can only deal with disk geometries of 1024 cylinders or less. Since virtually all modern disks have more than 1024 cylinders, most SCSI BIOS chips (which come on the SCSI controller card) and IDE BIOS (which is part of the rest of the PC BIOS) have an option (sometimes the default) to "translate" the real disk geometry into something that fits within MS-DOS' ability. However, not all BIOS chips "translate" the geometry in the same way. If you change your BIOS (either with a new motherboard or a new SCSI controller), and the new one uses a different "translated" geometry, you will be unable to load the second stage boot loader (and thus unable to load the kernel). (This is because the first stage boot loader contains a list of the blocks that comprise `/boot` in terms of the original "translated" geometry.) If you are using IDE disks, and you make changes to your BIOS settings, you can (unknowingly) change its translation also (most IDE BIOS offer 3 different translations.) To fix your boot block so that you can boot normally, just put a boot floppy in your hard drive, and at the boot prompt, type "`b hd0a:bsd`" to force it to boot from the first hard disk (and not the floppy). Your machine should come up normally. You now need to update the first stage boot loader to see the new geometry (and re-write the boot block accordingly).

Our example will assume your boot disk is `sd0` (but for IDE it would be `wd0`, etc.):

```
# cd /usr/mdec; ./installboot /boot biosboot sd0
```

If `installboot` complains that it is unable to read the BIOS geometry, at the `boot>` prompt you may issue the "`machine diskinfo`" (or "`ma di`" for short) command to print the information you need. Feed the "heads" and "secs" values to

installboot's -h and -s flags, respectively, so that the modified installboot command is the following:

```
# cd /usr/mdec; ./installboot -h <heads> -s <secs> /boot biosboot sd0
```

If a newer version of bootblocks are required, you will need to compile these yourself. To do so simply.

```
# cd /sys/arch/i386/stand/
# make && make install
# cd /usr/mdec; cp ./boot /boot
# ./installboot /boot biosboot sd0 (or whatever device your hard disk is)
```

14.9 - Preparing for disaster: Backing up and Restoring from tape

Introduction:

If you plan on running what might be called a production server, it is advisable to have some form of backup in the event one of your fixed disk drives fails.

This information will assist you in using the standard [dump\(8\)/restore\(8\)](#) utilities provided with OpenBSD. A more advanced backup utility called "Amanda" is also available for backing up multiple servers to one tape drive. In most environments [dump\(8\)/restore\(8\)](#) is enough. However if you have a need to backup multiple machines to one tape, Amanda might be worth investigating in the future.

The device examples in this document are for a configuration that uses both SCSI disks and tape. In a production environment, SCSI disks are recommended over IDE due to the way in which they handle bad blocks. That is not to say this information is useless if you are using an IDE disk or other type of tape drive, your device names will simply differ slightly. For example sd0a would be wd0a in an IDE based system.

Backing up to tape:

Backing up to tape requires knowledge of where your file systems are mounted. You can determine how your filesystems are mounted using the "mount" command at your shell prompt. You should get output similar to this:

```
shell# mount
/dev/sd0a on / type ffs (local)
/dev/sd0h on /usr type ffs (local)
```

In this example, the root (/) filesystem resides physically on sd0a which indicates SCSI fixed disk 0, partition a. The /usr filesystem resides on sd0h, which indicates SCSI fixed disk 0, partition h.

Another example of a more advanced mount table might be:

```
shell# mount
/dev/sd0a on / type ffs (local)
/dev/sd0d on /var type ffs (local)
/dev/sd0e on /home type ffs (local)
/dev/sd0h on /usr type ffs (local)
```

In this more advanced example, the root (/) filesystem resides physically on sd0a. The /var filesystem resides on sd0d, the /home filesystem on sd0e and finally /usr on sd0h.

To backup your machine you will need to feed dump the name of each fixed disk partition. Here is an example of the commands needed to backup the simpler mount table listed above:

```
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
# mt -f /dev/rst0 rewind
```

For the more advanced mount table example, you would use something similar to:

```
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0d
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0e
# /sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
# mt -f /dev/rst0 rewind
```

You can review the dump man page to learn exactly what each command line switch does. Here is a brief description of the parameters used above:

- **0** - Perform a level 0 dump, get everything
- **a** - Attempt to automatically determine tape media length
- **u** - Update the file /etc/dumpdates to indicate when backup was last performed
- **f** - Which tape device to use (/dev/nrst0 in this case)

Finally which partition to backup (/dev/rsd0a, etc)

The mt command is used at the end to rewind the drive. Review the mt man page for more options (such as eject).

If you are unsure of your tape device name, use dmesg to locate it. An example tape drive entry in dmesg might appear similar to:

```
st0 at scsibus0 targ 5 lun 0:
```

You may have noticed that when backing up, the tape drive is accessed as device name "nrst0" instead of the "st0" name that is seen in dmesg. When you access st0 as nrst0 you are accessing the same physical tape drive but telling the drive to not rewind at the end of the job and access the device in raw mode. To back up multiple file systems to a single tape, be sure you use the non-rewind device, if you use a rewind device (rst0) to back up multiple file systems, you'll end up overwriting the prior filesystem with the next one dump tries to write to tape. You can find a more elaborate description of various tape drive devices in the dump man page.

If you wanted to write a small script called "backup", it might look something like this:

```
echo " Starting Full Backup..."
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0a
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0d
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0e
/sbin/dump -0au -f /dev/nrst0 /dev/rsd0h
echo
echo -n " Rewinding Drive, Please wait..."
mt -f /dev/rst0 rewind
echo "Done."
echo
```

If scheduled nightly backups are desired, [cron\(8\)](#) could be used to launch your backup script automatically.

It will also be helpful to document (on a scrap of paper) how large each file system needs to be. You can use df -h to determine how much space each partition is currently using. This will be handy when the drive fails and you need to recreate your partition table on the new drive.

Restoring your data will also help reduce fragmentation. To ensure you get all files, the best way of backing up is rebooting your system in single user mode. File systems do not need to be mounted to be backed up. Don't forget to mount root (/) r/w after rebooting in single user mode or your dump will fail when trying to write out dumpdates. Enter `bsd -s` at the boot> prompt for single user mode.

Viewing the contents of a dump tape:

After you've backed up your file systems for the first time, it would be a good idea to briefly test your tape and be sure the data on it is as you expect it should be.

You can use the following example to review a catalog of files on a dump tape:

```
# /sbin/restore -tvvs 1 -f /dev/rst0
```

This will cause a list of files that exist on the 1st partition of the dump tape to be listed. Following along from the above examples, 1 would be your root (/) file system.

To see what resides on the 2nd tape partition and send the output to a file, you would use a command similar to:

```
# /sbin/restore -tvvs 2 -f /dev/rst0 > /home/me/list.txt
```

If you have a mount table like the simple one, 2 would be /usr, if yours is a more advanced mount table 2 might be /var or another fs. The sequence number matches the order in which the file systems are written to tape.

Restoring from tape:

The example scenario listed below would be useful if your fixed drive has failed completely. In the event you want to restore a single file from tape, review the restore man page and pay attention to the interactive mode instructions.

If you have prepared properly, replacing a disk and restoring your data from tape can be a very quick process. The standard OpenBSD install/boot floppy already contains the required restore utility as well as the binaries required to partition and make your new drive bootable. In most cases, this floppy and your most recent dump tape is all you'll need to get back up and running.

After physically replacing the failed disk drive, the basic steps to restore your data are as follows:

- Boot from the OpenBSD install/boot floppy. At the menu selection, choose Shell. Write protect and insert your most recent back up tape into the drive.
- Using the [fdisk\(8\)](#) command, create a primary OpenBSD partition on this newly intalled drive. Example:

```
shell# fdisk -e sd0
```

See [fdisk FAQ](#) for more info.

- Using the disklabel command, recreate your OpenBSD partition table inside that primary OpenBSD partition you just created with fdisk. Example:

```
shell# disklabel -E sd0
```

Don't forget swap, see [disklabel FAQ](#) for more info)

- Use the newfs command to build a clean file system on each partition you created in the above step. Example:

```
shell# newfs /dev/rsd0a
shell# newfs /dev/rsd0h
```

- Mount your newly prepared root (/) file system on /mnt. Example:

```
shell# mount /dev/sd0a /mnt
```

- Change into that mounted root file system and start the restore process. Example:

```
shell# cd /mnt
shell# restore -rs 1 -f /dev/rst0
```

- You'll want this new disk to be bootable, use the following to write a new MBR to your drive. Example:

```
shell# fdisk -i sd0
```

- In addition to writing a new MBR to the drive, you will need to install boot blocks to boot from it. The following is a brief example:

```
shell# cp /usr/mdec/boot /mnt/boot
shell# /usr/mdec/installboot -v /mnt/boot /usr/mdec/biosboot sd0
```

- Your new root file system on the fixed disk should be ready enough so you can boot it and continue restoring the rest of your file systems. Since your operating system is not complete yet, be sure you boot back up with single user mode. At the shell prompt, issue the following commands to unmount and halt the system:

```
shell# umount /mnt
shell# halt
```

- Remove the install/boot floppy from the drive and reboot your system. At the OpenBSD boot> prompt, issue the following command:

```
boot> bsd -s
```

The bsd -s will cause the kernel to be started in single user mode which will only require a root (/) file system.

- Assuming you performed the above steps correctly and nothing has gone wrong you should end up at a prompt asking you for a shell path or press return. Press return to use sh. Next, you'll want to remount root in r/w mode as opposed to read only. Issue the following command:

```
shell# mount -u -w /
```

- Once you have remounted in r/w mode you can continue restoring your other file systems. Example:

```
(simple mount table)
shell# mount /dev/sd0h /usr; cd /usr; restore -rs 2 -f /dev/rst0
```

```
(more advanced mount table)
shell# mount /dev/sd0d /var; cd /var; restore -rs 2 -f /dev/rst0
shell# mount /dev/sd0e /home; cd /home; restore -rs 3 -f /dev/rst0
shell# mount /dev/sd0h /usr; cd /usr; restore -rs 4 -f /dev/rst0
```

You could use "**restore rvsf**" instead of just rsf to view names of objects as they are extracted from the dump set.

- Finally after you finish restoring all your other file systems to disk, reboot into multiuser mode. If everything went as planned your system will be back to the state it was in as of your most recent back up tape and ready to use again.

[\[Back to Main Index\]](#) [\[To Section 13.0 - IPsec\]](#)



www@openbsd.org

\$OpenBSD: faq14.html,v 1.21 2000/04/07 20:17:49 ericj Exp \$

and authentication!

13.2 - That's nice, but why do I want to use IPsec?

The internet protocol, IP, aka IPv4, does not inherently provide any protection to your transferred data. It does not even guarantee that the sender is who he says he is. IPsec tries to remedy this. These services are considered distinct, but the IPsec supports them in a uniform manner.

Confidentiality

Make sure it is hard for anyone but the receiver to understand what data has been communicated. You do not want anyone to see your passwords when logging into a remote machine over the Internet.

Integrity

Guarantee that the data does not get changed on the way. If you are on a line carrying invoicing data you probably want to know that the amounts and account numbers are correct and not altered while in-transit.

Authenticity

Sign your data so that others can see that it is really you that sent it. It is clearly nice to know that documents are not forged.

Replay protection

We need ways to ensure a transaction can only be carried out once unless we are authorized to repeat it. I.e. it should not be possible for someone to record a transaction, and then replaying it verbatim, in order to get an effect of multiple transactions being received by the peer. Consider the attacker has got to know what the traffic is all about by other means than cracking the encryption, and that the traffic causes events favourable for him, like depositing money into his account. We need to make sure he cannot just replay that traffic later. *WARNING: as per the standards specification, replay protection is not performed when using manual-keyed IPsec (e.g., when using [ipsecadm\(8\)](#)).*

13.3 - What are the protocols behind IPsec?

IPsec provides confidentiality, integrity, authenticity, and replay protection through two new protocols. These protocols are called AH, Authentication header, and ESP, Encapsulated security payload.

AH provides authentication, integrity, and replay protection (but not confidentiality). Its main difference with ESP is that AH also secures parts of the IP header of the packet (like the source/destination addresses).

ESP can provide authentication, integrity, replay protection, and confidentiality of the data (it secures everything in the packet that follows the header). Replay protection requires authentication and integrity (these two go always together). Confidentiality (encryption) can be used with or without authentication/integrity. Similarly, one could use authentication/integrity with or without confidentiality.

13.4 - On the wire format

The **Authentication Header** (AH) comes after the basic IP header and contains cryptographic hashes of the data and identification information. The hashes can also cover the invariant parts of the IP header itself. There are several different RFCs giving a choice of actual algorithms to use in the AH, however they all must follow the guidelines specified in [RFC2402](#).

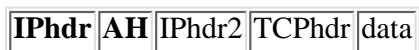
The **Encapsulating Security Payload** (ESP) header, allows for rewriting of the payload in encrypted form. The ESP header does not consider the fields of the IP header before it and therefore makes no guarantees about anything except the payload. The various types of ESP applicable must follow [RFC2406](#). An ESP header can also provide authentication for the payload, (but not the outer header).

An orthogonal (mostly) division of IPsec functionality is applied depending on whether the endpoint doing the IPsec encapsulation is the original source of the data or a gateway:

- **Transport** mode is used by a host that is generating the packets. In transport mode, the security headers are added before the transport layer (e.g. TCP, UDP) headers, before the IP header is prepended to the packet. In other words an AH added to the packet will cover the hashing of the TCP header and some fields of the end-to-end IP header, and an ESP header will cover the encryption of the TCP header and the data, but not the end-to-end IP header.
- **Tunnel** mode is used when the end-to-end IP header is already attached to the packet, and one of the ends of the secure connection is only a gateway. In this mode, the AH and ESP headers are used to cover the entire packet including the end-to-end header, and a new IP header is prepended to the packet that covers just the hop to the other end of the secure connection (though that may of course be several IP hops away).

IPsec secured links are defined in terms of **Security Associations** (SAs). Each SA is defined for a single unidirectional flow of data, and usually (ignoring multicast) from one single point to another, covering traffic distinguishable by some **unique selector**. All traffic flowing over a single SA is treated the same. Some traffic may be subject to several SAs, each of which applies some transform. Groups of SAs are called an SA **Bundle**. Incoming packets can be assigned to a particular SA by the three defining fields, (**Destination IP address, Security Parameter Index, security protocol**). SPI can be considered a cookie that is handed out by the receiver of the SA when the parameters of the connection are negotiated. The security protocol must be either AH or ESP. Since the IP address of the receiver is part of the triple, this is a guaranteed unique value. They can be found from the outer IP header and the first security header (which contains the SPI and the security protocol).

An example of a tunnel mode AH packet is:



An example of a transport mode AH packet is:



Because an ESP header cannot authenticate the outer IP header, it is useful to combine an AH and an ESP header to get the following:



This is called **Transport Adjacency**. The tunnelling version would look like:



However it is not specifically mentioned in the RFC. As with Transport adjacency, this would authenticate the entire packet except a few headers in the IP header and also encrypt the payload (seen in italics). When an AH and an ESP header are directly applied together like this, the order of the headers should be as shown. It is possible in tunnel mode, to do arbitrary recursive encapsulation so that order is not specified.

13.5 - Configuring IPsec

How the IPsec systems and gateways are configured is to some extent left to the designer, however the RFC has some strong recommendations as to how this should be implemented, so as to minimize confusion.

There are two administrative entities that control what happens to a packet. One is the **Security Association Database** (SAD, referred to as TDB or TDB table throughout OpenBSD's IPsec source code) and the other is the **Security Policy Database** (SPD).

They are similar in that given a number of selectors that describe some traffic, they will deliver an entry that describes the processing needed. However, the SPD is two steps removed from the actual processing: the SPD is used for outgoing packets, to decide what SAD entries should be used, and the SAD entries in turn describe the actual process and the parameters for it. The SPD entries specify the existing SAD entries to use (if it's a bundle there can be more than 1), but if there is not already a suitable one, it is used to create new ones. The fields of the SA being created can be taken either from the SPD entry or from the packet that initiated the creation.

Outgoing packets go from the SPD entry to the specific SA, to get encoding parameters. Incoming packets get to the correct SA directly using the SPI/DestIP/Proto triple, and from there get to the SPD entry.

The SPD can also specify what traffic should bypass IPSec and what should be dropped, so it must also be consulted for incoming non-IPSec traffic. SPD entries must be explicitly ordered as several might match a particular packet, and the processing must be reproducible.

The SPD can be thought of as similar to a packet filter where the actions decided upon are the activation of SA processes. Selectors can include src and dest address, port numbers if relevant, application and user IDs if available (only on host based transport SAs), hostnames, security sensitivity levels, protocols, etc.

A SAD entry would include:

- Dest IP address
- IPSec proto (SA or ESP)
- SPI (cookie)
- Sequence counter
- Seq O/F flag
- Anti-replay window info
- AH type and info
- ESP type and info
- Lifetime info
- Tunnel/transport mode flags
- Path MTU info

A SPD entry would contain:

- Pointer to active SAs
- Selector fields

Each SA can define one ESP header and one AH header. An IPSec session must have one or the other or both, but cannot be defined with neither - otherwise there would be no headers to specify the SPI to look up the SA. The RFC doesn't say what would happen if the AH and ESP headers disagree about the SPI value. One would presume this would imply multiple SAs in a bundle.

The SPD in OpenBSD is managed through the `ipsecadm flow` command. (You would only make changes to it if you were using manual keying.) SAD entries can be set manually with [ipsecadm\(8\)](#), however the IETF has also defined automatic mechanisms for initialization of sessions and such things as key exchange. OpenBSD implements both Photuris ([RFC2522](#), and [RFC2523](#)) and ISAKMP automatic key exchange ([RFC2407](#), [RFC2408](#), and [RFC2409](#)) in the [photurisd\(8\)](#) and [isakmpd\(8\)](#) daemons.

13.6 - How do I setup IPSec with manual keying?

Manual keying is the easiest way to get started with IPSec. You can setup encryption between networks, to create VPNs using this method. After you have read this section, you may want to investigate using [/usr/share/ipsec/rc.vpn](#) to set this up for you automatically.

First, you need to turn on IP AH and IP ESP options in the OpenBSD kernel (if you are only using ESP, such as with the `rc.vpn` script, or as with the example below, then *you do not need to enable AH*. In fact, there may even be security concerns related to enabling AH if you are not using it).

There is a nice `sysctl` to enable these protocols.

```
# sysctl -w net.inet.esp.enable=1
net.inet.esp.enable: 0 -> 1
# sysctl -w net.inet.ah.enable=1
net.inet.ah.enable: 0 -> 1
```

You can edit `/etc/sysctl.conf` to turn these on at boot time. You need to remove the `#` mark from in front of `net.inet.esp.enable` and/or `net.inet.ah.enable` (depending on which you plan to use) and make sure they are set to 1.

You also need to generate your manual keys. Since the security of the VPN is based on these keys being unguessable, it is very important that the keys be chosen using a strong random source. One practical method of generating them is by using the [random\(4\)](#) device. To produce 160 bits of randomness, for example, do:

```
# dd if=/dev/urandom bs=1024 count=1 | sha1
```

The number of bits produced is important. Different cipher types may require different sized keys.

Cipher	Key Length
DES	56 bits
3DES	168 bits
BLF	Variable (160 bits recommended)
CAST	Variable (160 bits recommended)
SKIPJACK	80 bits

Now, you need to setup SAs, or Security Associations. A Security Association is a combination of your IP addresses, an SPI, and your security protocol (AH and/or ESP). The IP addresses are both your own and that of your destination. The SPI, or Security Parameter Index, is a number that OpenBSD uses to classify different SAs.

These examples only use ESP to encrypt your traffic. ESP includes authentication of the contained encrypted data, but does not authenticate the surrounding IP header, as AH would. This "limited authentication" is nevertheless quite sufficient in most cases, especially for ESP in a tunnel environment.

```
# ipsecadm new esp -spi SPI_OUT -src MY_EXTERNAL_IP -dst PEER_EXTERNAL_IP
-forcetunnel -enc blf -auth sha1 -key ENC_KEY -authkey AUTH_KEY
```

Let's put this into practice with two routers, 192.168.5.1 and 192.168.25.9.

On Host 192.168.5.1:

```
# ipsecadm new esp -spi 1000 -src 192.168.5.1 -dst 192.168.25.9 -forcetunnel -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -spi 1001 -dst 192.168.5.1 -src 192.168.25.9 -forcetunnel -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
```

On Host 192.168.25.9:

```
# ipsecadm new esp -spi 1001 -src 192.168.25.9 -dst 192.168.5.1 -forcetunnel -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -spi 1000 -dst 192.168.25.9 -src 192.168.5.1 -forcetunnel -enc
blf -auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
6a20367e21c66e5a40739db293cf2ef2a4e6659f
```

Notice that the SPIs are different. See [On the wire format](#) for a complete description of what the SPI is and where it is used.

Now that you have your Security Associations in place, set up your flows.

On 192.168.5.1:

So, right here, **two** flows will be created, one the local source address, which covers all packets originating from the local host to the destination, as well as a flow from the destination back to the local host.

```
# ipsecadm flow -proto esp -dst 192.168.25.9 -spi 1000 -addr 192.168.5.1
255.255.255.255 192.168.25.9 255.255.255.255
```

On 192.168.25.9:

```
# ipsecadm flow -proto esp -dst 192.168.5.1 -spi 1001 -addr 192.168.25.9
255.255.255.255 192.168.5.1 255.255.255.255
```

If you want less overhead on your Host-to-Host VPNs, creating the SPI without `-forcetunnel` will let you use transport

mode (whereas, `-forcetunnel` makes sure all of the IP packet, including the IP header, are encapsulated by SPI). If either the source or destination is a network, you will have to use tunnel mode. Creating an SA to and/or from a network will automatically ensure tunnel mode SPIs are being created.

This is a simple way to start using IPSec.

You can use IPSec to tunnel private IP address spaces over the Internet. Here is a good example... We want to tunnel 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.2.0/24 and 208.1.5.0/24 which are behind 208.2.2.2. These examples were generated using the [rc.vpn](#) script.

As you can see, when you are using manual keying with IPSec, you have to specify **exactly** what you want done. It won't guess for you. Look at these examples...

On 208.1.1.1:

First, set up the security associations (SAs):

(This sets up the SPIs, encryption methods, and keys.)

```
# ipsecadm new esp -src 208.1.1.1 -dst 208.2.2.2 -forcetunnel -spi 1001 -enc blf
-auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -src 208.2.2.2 -dst 208.1.1.1 -forcetunnel -spi 1000 -enc blf
-auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
```

Next, setup a flow from 208.1.1.1 to 208.2.2.2

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.2.2.2 255.255.255.255
```

Next, setup a flow from 208.1.2.0/24, which is behind 208.2.2.2, to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.1.2.0 255.255.255.0
```

Next, setup a flow from 208.1.5.0/24, which is behind 208.2.2.2, to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.1.5.0 255.255.255.0
```

Now, setup a flow from 208.1.2.0/24, which is behind 208.2.2.2 to the router 208.1.1.1.

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.1.2.0 255.255.255.0
```

OK, setup a flow from 208.1.5.0/24, which is behind 208.2.2.2, to the router 208.1.1.1

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 208.1.1.1
255.255.255.255 208.1.5.0 255.255.255.0
```

Finally, setup a flow from the router 208.2.2.2 to 192.168.99.0/24

```
# ipsecadm flow -proto esp -dst 208.2.2.2 -spi 1001 -addr 192.168.99.0
255.255.255.0 208.2.2.2 255.255.255.255
```

On 208.2.2.2:

Same as before, we setup the SAs...

```
# ipsecadm new esp -src 208.2.2.2 -dst 208.1.1.1 -forcetunnel -spi 1000 -enc blf
-auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
# ipsecadm new esp -src 208.1.1.1 -dst 208.2.2.2 -forcetunnel -spi 1001 -enc blf
-auth sha1 -key 7762d8707255d974168cbb1d274f8bed4cbd3364 -authkey
67e21c66e5a40739db293cf2ef2a4e6659f
```

Now, this is the reverse side... Setup a flow from the router 208.2.2.2 to 208.1.1.1

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.2.2.2
255.255.255.255 208.1.1.1 255.255.255.255
```

Setup a flow from the network 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.2.0/24

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.2.0 255.255.255.0
192.168.99.0 255.255.255.0
```

Setup a flow from the network 192.168.99.0/24, which is behind 208.1.1.1, to 208.1.5.0/24

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.5.0 255.255.255.0
192.168.99.0 255.255.255.0
```

Now, setup a flow from 192.169.99.0/24, which is behind 208.1.1.1, to the router 208.2.2.2

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.2.2.2
255.255.255.255 192.168.99.0 255.255.255.0
```

We're almost done... Two flows left to get 208.1.2.0/24 and 208.1.5.0/24 from the router 208.2.2.2 to the router 208.1.1.1.

```
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.2.0 255.255.255.0
208.2.2.2 255.255.255.255 -ingress
# ipsecadm flow -proto esp -dst 208.1.1.1 -spi 1000 -addr 208.1.5.0 255.255.255.0
208.2.2.2 255.255.255.255 -ingress
```

If you have been using ipsecadm, and you want to get rid of any work that you've done, and start from scratch, do

```
# ipsecadm flush
```

This will flush all IPsec info (SPIs, flows, routing entries) from your system.

13.7 - How do I setup photurisd?

Photuris is not widely used and is still considered experimental as far as the RFC status is concerned. However, many people have used it with OpenBSD.

To setup your photurisd, first edit /etc/photuris/secrets.conf on each host with photurisd.

```
bsd# cat /etc/photuris/secrets.conf
# Accepted keywords are:
# identity local "id" "secret"
# identity pair local "receivedid" "myid" "secret"
# identity remote "id" "secret"
# identity lookup "tag" username
# Simple
identity local "Default" "This should be changed."
identity remote "Default" "This should be changed."
```

Change "This should be changed." to a key of your choice on the local config, and another key of your choice on the remote config. (Use the same config on your remote box but swap "local" and "remote" so that it sees itself as the local key.) Note that these keys will be replaced in a future version of photurisd that will carry out its initial key exchange with public keys.

Make sure net.inet.ah.enable is set to 1.

```
bsd# sysctl -w net.inet.ah.enable=1
net.inet.ah.enable: 0 -> 1
```

And run startkey.

```
bsd# startkey dst=remote.host
```

Now, run tcpdump to verify that your packets are being encrypted with AH. (Run a ping in another window or session to generate traffic.)

```
bsd# tcpdump proto ah
```

You can also try using tcpdump by the host address if you aren't getting anything.

```
bsd# tcpdump host remote.host
```

You can make photurisd automatically set the source and destination host or networks in /etc/photuris/photuris.startup

13.8 - How do I setup isakmpd?

If you are thinking about VPNs or other traditional applications of IPsec, you probably are going to use ISAKMP. Some commercial implementations of IPsec do not provide any manual keying ability, instead they require you to use some form of ISAKMP.

13.8.1 - What is isakmpd?

ISAKMP (sometimes referred to as IKE, or Internet Key Exchange) is the key exchange mechanism for the VPN. It meets security concerns using the methods mentioned in RFC 2407, RFC 2408 and RFC 2409. ISAKMP manages the exchange of cryptographic keys that you would normally have to manage with [ipsecadm\(8\)](#). It employs a two-phase process for establishing the IPsec parameters between two IPsec nodes.

Phase 1 - The two ISAKMP peers establish a secure, authenticated channel upon which to communicate between two daemons. This establishes a Security Association (SA) between both hosts. **Main Mode** and **Aggressive Mode** are the methods used to establish this channel. Main Mode sends the various authentication information in a certain sequence, providing identity protection. Aggressive Mode does not provide identity protection because all of the authentication information is sent at the same time. Aggressive mode should only be used in such cases where network bandwidth is of concern.

Phase 2 - Security Associations are negotiated on behalf of IPsec. Phase 2 is establishes tunnels or endpoint SAs between IPsec hosts. **Quick Mode** is used in Phase 2 because there is not need to repeat a full authentication because Phase 1 has already established the SAs.

In brief, Phase 1 is used to get a secure channel in which to do the (quicker) phase 2 setups. There can be multiple phase 2 setups within the same phase 1 channel. Phase 2 is used to setup the actual tunnels. In Phase 1, your IPsec nodes establish a connection where they exchange authentication (Either a X509 certificate or a pre-shared secret). This allows each end to make sure the other end is authenticated. Phase 2 is an exchange of keys to determine how the data between the two will be encrypted.

13.8.2 - How do I get started with isakmpd?

By default, OpenBSD comes with the proper binaries for ISAKMP and the IPsec stack. Unfortunately, the same cannot be said for the sample files. To retrieve them, you need to grab: /usr/src/sbin/isakmpd/samples/VPN-east.conf and /usr/src/sbin/isakmpd/samples/policy from the source tree. You can either use your CD (if you have one), [cvswweb](#), or the [command line CVS client](#). (Cvswweb has [VPN-east.conf](#) and [policy](#) readily available). For the purposes of this example, copy *policy* to /etc/isakmpd.policy. Copy *VPN-east.conf* to /etc/isakmpd/isakmpd.conf. Here we attempt to show you how to setup a VPN (tunnel). If you want to use isakmpd between single hosts, there are other configuration files in the *samples* directory. The manual pages have detailed information.. Don't forget [isakmpd.conf\(5\)](#) and [isakmpd.policy\(5\)](#).

Your first step is to turn on esp. The [top of section 13.6](#) tells you how to do this both for run-time and boot. Next, you need to edit /etc/isakmpd.policy. This file tells ISAKMP who can access IPsec. In this scenario, the policy file states that anybody who sends data using Encapsulate Security Payload(ESP), and has authenticated with the passphrase mekmitasdigoat (or whatever passphrase you determine), is allowed to communicate with isakmpd. You can modify this file to let ISAKMP know that we only want to allow data signed with certain digital certificates or using a certain encryption transform. You could also allow anybody to access IPsec. This is only recommended for testing. To do this, edit your policy file to contain only the following lines:

```
KeyNote-Version: 2
Authorizer: "POLICY"
```

The same policy file contains two lines that start with the \$ character. You need to remove these lines before using it, they are only for cvs.

A more useful policy file for this example looks like this:

```
KeyNote-Version: 2
Comment: This policy accepts ESP SAs from a remote that uses the right password
```

```

Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" -> "true";

```

Implementing this will give you a basic VPN (tunnel) using ESP only. On host A, edit `/etc/isakmpd/isakmpd.conf`. The 249.2.2.2 sample IP address should be replaced with the external IP address of host A.

```

[General]
Retransmits=          5
Exchange-max-time=    120
Listen-on=            249.2.2.2

```

Do similar for `isakmpd.conf` on host B. 249.3.3.3 represents the external IP address for host B.

```

[General]
Retransmits=          5
Exchange-max-time=    120
Listen-on=            249.3.3.3

```

This is where you can setup the variables that will affect the main behaviour of `isakmpd`. It is okay to use the defaults here. The **Listen-on=** value specifies the IP that `isakmpd` should listen on. Only the Internet IP of your gateway is necessary. If you have multiple external interfaces on your gateway, you could list which interfaces you want to listening on by entering them using a comma separated list.

Next, on host A, edit `isakmpd.conf` again.

```

[Phase 1]
249.3.3.3=            HostB

```

On host B:

```

[Phase 1]
249.2.2.2=            HostA

```

This section describes the IP addresses to accept in order to negotiate the phase 1 connection. Its value points to the section below (Remember that phase 1 simply authenticates the remote peer to make sure they are who they say they are). You can list multiple peers with additional lines in the format of `IP_Address= <PEER-NAME>`.

Next, on host A:

```

[Phase 2]
Connections=          HostA-HostB

```

On host B:

```

[Phase 2]
Connections=          HostB-HostA

```

This describes Phase 2 of the connection. This is the phase that determines what protocols the two peers will use to communicate.

The **Connections=** tag refers to the section below. It initiates the requirements or the accepted methods to set up Phase 2. This also tells `ISAKMPD` which connections to initiate once started. Note that you can have multiple sections as illustrated below if you are to connect with multiple peer hosts.

If you do not have the IP address of the remote host, you can specify a `Default=` that points to a section describing a generic entry that will be referenced by any incoming IP that is not listed in the **Connections=** tag.

On host A:

```

[HostB]
Phase=                1
Transport=            udp
Local-address=        249.2.2.2
Address=              249.3.3.3
Configuration=        Default-main-mode
Authentication=       mekmitasdigoat

```

```
#Flags=
```

On host B:

```
[HostA]
Phase=                1
Transport=            udp
Local-address=        249.3.3.3
Address=              249.2.2.2
Configuration=        Default-main-mode
Authentication=        mekmitasdigoat
#Flags=
```

These represent the sections referred to by the Phase 1 section above. They each describe the requirements that the peer gateway must fulfil in order to proceed to Phase 2. There are many other options here but the ones mentioned above are the minimum requirements.

- **Phase=1** is required because the ISAKMPD code uses the same procedures to process Phase 1 and Phase 2. It must be **1** or nothing will work.
- **Transport=** gives you different possibilities for different peers. It's suggested that `udp` be used here so we'll leave it at that. Please note that some peers may be behind a firewall that doesn't let UDP traffic through. Obviously, this needs to be determined before setup.
- **Local-address** is the destination address that the incoming packets point to. In some cases, you can be listening on different Interfaces for Phase 1 connections. In this example, there is only 1 interface listening, therefore this is the IP of the listening interface on this peer.
- **Address=** is the address that points to the source IP of the incoming packets. This usually points to the peer gateway. This needs further explanation, because the source IP address of the peer may be unknown!
- **Configuration=** points to the section below. You can specify multiple sections like this. We use the default one specified by the sample file.
- **Authentication=** is the pre-shared secret to be used for this particular peer. It is more or less a passphrase that each peer uses. This passphrase gets passed to policy to verify whether this peer is allowed to use IPSEC with this host. If you change this phrase, you must also change it in the policy file because the sample file provides for this passphrase. If you decided to go with a minimum policy file then you can specify whatever you want here.
- **Flags=** is not currently being used. The RFCs leave room for extra options to be specified for phase 1.

There are other tags here that will allow for other options to be set. Refer to [isakmpd.conf\(5\)](#) for descriptions.

On Host A:

```
[HostA-HostB]
Phase=                2
ISAKMP-peer=          HostB
Configuration=        Default-quick-mode
Local-ID=              Net-A
Remote-ID=             Net-B
```

On Host B:

```
[HostB-HostA]
Phase=                2
ISAKMP-peer=          HostA
Configuration=        Default-quick-mode
Local-ID=              Net-B
Remote-ID=             Net-A
```

These represent the sections referred to by the Phase 2 section above. They are the individual settings that ISAKMPD must use to talk between the two gateways for the particular connection.

- **Phase=2** is required because ISAKMPD code uses the same functions to authenticate Phase 1 and Phase 2. This is required for the VPN to work.
- **ISAKMPD-Peer=** is the name of Host section above. This means that we are talking to that particular peer to establish a

Phase 2 connection. This is provided because you can have multiple sections to describe isakmp peers and connections.

- **Configuration=** refers to the section below that describes the standards by which this host and the particular peer for this connection must abide.
- **Local-ID=** refers to an IPsec-ID section below that describes our Private Network to the peer gateway. This is the portion that is passed so that the other gateway can set up the proper routing table that will transfer data over the VPN to our network.
- **Remote-ID=** refers to an IPsec-ID section below that describes what is supposed to be the remote Private Network to our host. This portion is interpreted to set up the proper routing tables that will transfer data from our Private Network over the VPN to the remote Private Network.

There is another tag that is supported here called **Flags=**. If you require this tag, read [isakmpd.conf\(5\)](#).

This is the IPsec-ID section. These entries need to exist in the `isakmpd.conf` files for both Host A and Host B. This example will setup 192.168.1.0/255.255.255.0 for Host A (which was connected to Net-A above) and 192.168.20.0/255.255.255.0 for Host B (Net-B above).

```
[Net-A]
ID-type=          IPV4_ADDR_SUBNET
Network=          192.168.1.0
Netmask=          255.255.255.0

[Net-B]
ID-type=          IPV4_ADDR_SUBNET
Network=          192.168.20.0
Netmask=          255.255.255.0
```

These two sections are in the **conf** file of each host. They are the sections referenced by the **Local-ID** and **Remote-ID** identifiers. They describe the routes that should be set up to allow traffic from one private network to another. **ID-type=** can be **IPV4_ADDR_SUBNET** or **IPV4_ADDR** (RFC2708 mentions more possible values. Currently only IPv4 is supported in the OpenBSD implementation. IPv6 may be supported in OpenBSD-current.)

Now, on both hosts, the sample file should read:

```
[Default-main-mode]
DOI=                IPSEC
EXCHANGE_TYPE=     ID_PROT
Transforms=        3DES-SHA
```

This section describes the requirements for the encryption methods of Phase 1 connections. The name reflects the value of **Configuration=** variable. As we can see here, we are stating our Domain of Interest which is IPSEC. The **EXCHANGE_TYPE** variable is set to **ID_PROT** for Phase 1, which identifies the protocols to be covered by this Authentication. **Transforms=** is the transform required (or assigned) for this exchange. In this case, this points to the section below in the configuration file that says we are receiving a packet encrypted with 3DES and a checksum verifiable with SHA. There are a bunch of different transforms defined inside the sample **VPN-east.conf**. These are provided because 3DES and SHA are not always supported across different platforms. For OpenBSD there should be no reason to change this for a basic setup. Feel free to create multiples of this section and change the transform. The only requirement is that you change the **Configuration=** variable.

```
[Default-quick-mode]
DOI=                IPSEC
EXCHANGE_TYPE=     QUICK_MODE
Suites=             QM-ESP-3DES-SHA-PFS-SUITE , QM-ESP-DES-MD5-PFS-SUITE
```

This section describes the requirements for the encryption of the data to be sent through the VPN and is referred to by **Configuration** above. Note the difference between this section and the Phase 1 equivalent just above is that the **EXCHANGE_TYPE** is **QUICK_MODE**. This is always the case for Phase 2. **Suites=** points to a IPsec Suite section describing the different encryption schemes available between the two hosts. There is much more to be said about ISAKMP and IPsec. By using the above basic descriptions you should be able to create a simple but solid VPN that cares and feeds itself. This is the bare *minimum* **isakmpd.conf** for both hosts here.

13.8.3 - Starting isakmpd

You may wish to use

```
# isakmpd -d -DA=99
```

the first time you decide to run this daemon. The daemon will not be running in daemon mode but as a regular process. It will log everything to your terminal. To stop isakmpd and flush the routes, you need to kill the isakmpd process on each node, and run `ipsecadm flush`.

13.9 - How do I use isakmpd with X.509 certificates?

Setting up isakmpd to use certificates instead of pre-shared keys is not really that much harder in a big network with many untrusted peers than it would be with a small network. It may actually simplify configuration, and more importantly, key management.

Generating certificates.

There is a good description of how to generate keys and certificates in the [README.PKI](#) file in the isakmpd source directory. You need to have a CA key, a corresponding CA X.509 certificate, one private key for each computer on the network that will use isakmpd and one X.509 certificate for each such key.

The X.509 certificates need to have a Subject Alternative Name (SubjectAltName) extension describing the certificate holder. How to set a SubjectAltName extension using `certpatch` for a certificate is also described in `README.PKI` for the case of setting an IP address as the SubjectAltName. Using an IP address here is also the default behaviour for isakmpd.

`Certpatch` also support using either a FQDN (Fully Qualified Domain Name) or a UFQDN (User FQDN). An example of an FQDN might be `www.openbsd.org`, an example of an UFQDN would be an email address. Something like `Jorgen.Granstam@abc.se` for example.

In this howto document I am going to use FQDNs as SubjectAltNames. Using IP addresses would be a bit easier since that is the default behaviour of isakmpd but it is not much of a difference as we soon will see.

To insert an FQDN SubjectAltName into a certificate one would do something like this:

```
$ /usr/sbin/certpatch -t fqdn -i home.mysite.se -k ca.key originalcert.crt
newcert.crt
```

Here the `ca.key` is the private key of the CA, thus this can only be done by whoever has access to the CA private key. The (fictional) `home.mysite.se` is the FQDN to be inserted into the certificate. The `originalcert.crt` and `newcert.crt` filenames may be the same name in which case the original file will be overwritten by the new modified certificate.

Put the keys and certificates in the directories as described at the end of `README.PKI`. The CA key (`ca.key`) should be kept in some safe place if the keys are really to be seriously used.

Configuration of isakmpd

Lets now look at the `/etc/isakmpd/isakmpd.conf` configuration file. It was originally taken from the example file in [isakmpd.conf\(5\)](#) but have been heavily modified. I will also use a much shorter file here than the example file in the man-page, I have removed from that file most parts that are not needed in this setup. In part to simplify the understanding of this setup. I have also added some commenting (some comments are left as in the `isakmpd.conf(5)`) and changed some names. None of the domain names used here exists as far as I know.

Actually since everyone who reads this already have a working configuration for the preshared keys case (see previous section) there won't be many surprises in this file. I won't explain this in every detail, check `isakmpd.conf(5)` for descriptions of the parts I don't comment on.

Let's assume our setup looks something like this

```

one.mysite.se                one.worksite.se
192.168.1.2--+      10.0.0.1====/=====10.0.0.2      +--192.168.2.2
                    | gw.mysite.se          gw.worksite.se |
                    +--192.168.1.1          192.168.2.1---+
two.mysite.se |                | two.worksite.se
192.168.1.3--+                +--192.168.2.3

```

That is, two networks that should be connected using an IPSec tunnel over an otherwise insecure network. Ignore the fact that I am using IP addresses reserved for private Internets here (RFC1918), I have to use something. I won't explain how to use isakmpd in combination with NAT or similar (because I haven't tried that myself).

Now, let's look at the configuration file. This is the file for the security gateway gw.mysite.se:

```

# *****
# ***** Start of the gw.mysite.se isakmpd.conf *****
# *****

# A configuration sample for the isakmpd ISAKMP/Oakley (aka IKE) daemon.
[General]
Policy-File=          /etc/isakmpd/policy
Retransmits=         5
Exchange-max-time=   120
Listen-on=           10.0.0.1

# The name work-gw here is used just as a section name and a tag for
# use in this configuration file below and need not actually be the
# real hostname or domain name of the peer (but it could be). The IP
# address however needs to be correct. Phase 1, as you might already
# know, is to negotiate an ISAKMP security association (SA). There
# should of course be one IP and name for each peer we want to
# communicate with.
[Phase 1]
10.0.0.2=             work-gw

# Now phase 2 is negotiating IPSec SAs. As in phase 1, the name here
# is a section name to be used later. Actually, it can be a comma
# separated list of section names here. Thus if traffic from many
# networks (or individual hosts) should be forwarded through this
# tunnel, more sectionnames would be added (and of course corresponding
# new sections further down).
[Phase 2]
Connections=         work-gw-my-gw

# Now, here are some parameters for the ISAKMP SA negotiations. Almost
# self documenting. The section name is from [Phase 1] above. The most
# interesting tag might be the ID tag. The ID tag is set to the name
# of the section where the identity information about this host that
# will be presented to connecting peers, can be found. If the ID tag
# is not available, isakmpd will assume that it will identify itself
# using the IP address. You might also notice that there is no longer
# any authentication tag here in this configuration. The authentication
# data is currently used only in the preshared key case.

```

13.0 - Using IPSec

```
[work-gw]
Phase=                1
Transport=            udp
Local-address=        10.0.0.1          # Local address
Address=              10.0.0.2          # Peer address
ID=                  my-ID
Configuration=        Default-main-mode
```

```
# This is the identity data. ID-type may also be IPV4_ADDR (the
# default), IPV4_ADDR_SUBNET or UFQDN. The Name tag is used for
# FQDN and UFQDN, for IPV4_ADDR an Address tag would be used instead.
# For IPV4_ADDR_SUBNET a Network and a Netmask tag would be used.
```

```
[my-ID]
ID-type=              FQDN
Name=                 gw.mysite.se
```

```
# This is the section for the IPSec connection. The section name is
# from the list in the [Phase 2] section above. The ISAKMP-peer is,
# of course, the tag of our peer from section [Phase 1] above. The
# Local-ID and Remote-ID tags should be section names describing which
# packages should be forwarded over the IPSec tunnel to the remote
# network.
```

```
[work-gw-my-gw]
Phase=                2
ISAKMP-peer=          work-gw
Configuration=        Default-quick-mode
Local-ID=              Net-west
Remote-ID=             Net-east
```

```
# Any packet originating from a computer on the network described
# here...
```

```
[Net-west]
ID-type=              IPV4_ADDR_SUBNET
Network=              192.168.1.0
Netmask=              255.255.255.0
```

```
# ... and with a destination matching the network described here,
# will be encrypted and forwarded over the IPSec tunnel to the remote
# system.
```

```
[Net-east]
ID-type=              IPV4_ADDR_SUBNET
Network=              192.168.2.0
Netmask=              255.255.255.0
```

```
# Main mode descriptions
```

```
# Here are the data for main mode. Using DES here for real purposes
# is not very smart since DES is no longer considered a secure
# encryption algorithm. 3DES is generally considered to have much better
# security since it has enough bits in the key to be considered secure.
# Transforms is a list of tags describing main mode transforms. In
# this example we have only one.
```

```
[Default-main-mode]
```

13.0 - Using IPsec

```
DOI= IPSEC
EXCHANGE_TYPE= ID_PROT
Transforms= 3DES-MD5
```

```
# Certificates stored in PEM format
# This is important when using certificates. The CA certificates should
# be in the CA-directory (but not the CA private key ofcourse).
# The Cert-directory should have at least the certificate for the
# local host but other certificates are also allowed. The private key
# should be the private key of the local host.
```

```
[X509-certificates]
```

```
CA-directory= /etc/isakmpd/ca/
Cert-directory= /etc/isakmpd/certs/
Private-key= /etc/isakmpd/private/local.key
```

```
# Main mode transforms
#####
```

```
# Here is our main mode transform. The important thing here is to use
# RSA_SIG as authentication method when using certificates. It is the
# only method supported when using certificates so far. Commercial
# entities in the US will thus have to wait until September 2000 to
# use this due to the RSA patent. Luckily, I am not living in the US.
# Also important is the GROUP_DESCRIPTION tag. It must match the
# GROUP_DESCRIPTION tag in the Quick mode transforms further down.
# The Life tag here could possibly be modified. The LIFE_60_SECS might
# be shorter than necessary for normal use.
```

```
[3DES-MD5]
```

```
ENCRYPTION_ALGORITHM= 3DES_CBC
HASH_ALGORITHM= MD5
AUTHENTICATION_METHOD= RSA_SIG
GROUP_DESCRIPTION= MODP_1024
Life= LIFE_60_SECS,LIFE_1000_KB
```

```
# Quick mode description
#####
```

```
[Default-quick-mode]
```

```
DOI= IPSEC
EXCHANGE_TYPE= QUICK_MODE
Suites= QM-ESP-3DES-MD5-PFS-SUITE
```

```
# Quick mode protection suites
#####
# 3DES
```

```
[QM-ESP-3DES-MD5-PFS-SUITE]
```

```
Protocols= QM-ESP-3DES-MD5-PFS
```

```
# 3DES
```

```
[QM-ESP-3DES-MD5-PFS]
```

```
PROTOCOL_ID= IPSEC_ESP
Transforms= QM-ESP-3DES-MD5-PFS-XF
```

```

# Quick mode transforms

# Don't forget. The GROUP_DESCRIPTION must match the GROUP_DESCRIPTION
# in main mode above. For forwarding packets between two networks (or
# from a host to a network) we use TUNNEL mode. Between two hosts we
# may also use TRANSPORT mode instead.
[QM-ESP-3DES-MD5-PFS-XF]
TRANSFORM_ID=          3DES
ENCAPSULATION_MODE=    TUNNEL
AUTHENTICATION_ALGORITHM=      HMAC_MD5
GROUP_DESCRIPTION=     MODP_1024
Life=                  LIFE_60_SECS

# As we know from the isakmpd.config manpage the LIFE_DURATION here is
# an offer value (60), a minimum acceptable value (45) and a maximum
# acceptable value. The isakmpd.conf example has this set to
# 600,450/720 instead. That might be a better value for normal use.
[LIFE_60_SECS]
LIFE_TYPE=              SECONDS
LIFE_DURATION=          60,45:72

[LIFE_1000_KB]
LIFE_TYPE=              KILOBYTES
LIFE_DURATION=          1000,768:1536

# *****
# ***** End of the gw.mysite.se isakmpd.conf *****
# *****

```

So far the configuration for the local system. The remote system is configured just the same way only opposite. Thus only the first part of the isakmpd.conf file differs. Let's just look at that first part of the isakmpd.conf file for the security gateway gw.worksite.se:

```

# *****
# ***** Start of the gw.worksite.se isakmpd.conf *****
# *****

[General]
Policy-File=           /etc/isakmpd/policy
Retransmits=           5
Exchange-max-time=     120
Listen-on=             10.0.0.2

[Phase 1]
10.0.0.1=              my-gw

[Phase 2]
Connections=           work-gw-my-gw

[my-gw]
Phase=                 1
Transport=             udp
Local-address=         10.0.0.2           # Local address
Address=               10.0.0.1         # Peer address
ID=                   work-ID

```

```

Configuration=          Default-main-mode

[work-ID]
ID-type=                FQDN
Name=                   gw.worksite.se

[work-gw-my-gw]
Phase=                  2
ISAKMP-peer=           my-gw
Configuration=         Default-quick-mode
Local-ID=               Net-east
Remote-ID=              Net-west

# *****
# ***** .. to be continued *****
# *****

```

Now that wasn't so hard, just a bit boring to read perhaps. A slightly more interesting part next.

The policy file.

Actually, the policy file might be slightly confusing for anyone who has not used it before, especially if things doesn't work as expected. The man-page [isakmpd.policy\(5\)](#) is not really that bad. It might perhaps be a little bit unclear in some parts but generally it's good.

The simplest possible working policy file would contain just a single line:

```
authorizer: "POLICY"
```

This basically means that there is no policy limitations on who would be allowed to connect. Thus not a very secure setup. The authorizer tag here means the one who has the authorization to decide the policy. The special authorizer "POLICY" has the ultimate and unlimited authority on policy. Any other authorizer must first be authorized by "POLICY" to have any authority here.

There can also be a set of conditions for what is allowed. The following policy thus would mean that only someone using the ESP protocol with some real encryption would be authorized (oh well, someone using DES would also be authorized here although DES could almost be considered snakeoil today, it is left as an exercise for the reader to change this policy into not allowing DES either). Note that anyone who does encryptions with ESP would still be allowed.

```

authorizer: "POLICY"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";

```

It is also possible to "sublicense" authority to someone else (might be one or more entities). The simple case would be the preshared key case. In that case, anyone who knows the preshared passphrase is authorized. Thus:

```

authorizer: "POLICY"
licensees: "passphrase:something really secret"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";

```

This would authorize anyone who knows this passphrase to connect and comply with the conditions (but remember that the passphrase must also be set in the Authentication tag in isakmpd.conf).

Nothing difficult so far. Now to the interesting stuff. First there can be many licensees, although all must be authorized by "POLICY". Further, authorized licensees can sublicense to other licensees. A licensee can be just a string in case it is further described in the policy file:

13.0 - Using IPsec

```
authorizer: "POLICY"  
licensees: "subpolicyAH" || "subpolicyESP"  
conditions: app_domain == "IPsec policy" -> "true";
```

```
authorizer: "subpolicyESP"  
licensees: "passphrase:something more secret"  
conditions: esp_present == "yes" -> "true";
```

```
authorizer: "subpolicyAH"  
licensees: "passphrase:something really secret"  
conditions: ah_present == "yes" -> "true";
```

And now to what everyone has been waiting for. Policy can also be sublicensed or delegated to a key. In this case it is usually a X.509 certificate. The simple use of certificates would be to use them like the passphrases. Just insert individual users certificates in the policy file:

```
keynote-version: 2  
comment: This is an example of a policy delegating to a key.  
authorizer: "POLICY"  
licensees: "x509-base64:\n  
MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\  
BAMTCUllRUxBQIBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\  
CzA          This is would be a user certificate          AQEB\  
BQA          IUuz\  
eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\  
ct+016zUBP/cQMml+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjPui3AgMB\  
AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\  
iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaa\  
NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\  
yiXHSU8="\  
conditions: app_domain == "IPsec policy" &&  
             esp_present == "yes" &&  
             esp_enc_alg != "null" -> "true";
```

Now, this is obviously a stupid idea if there are a lot of users. The certificates that isakmpd reads from the CA- and Certificate directories, and the certificates received from the peer is converted into pseudo credentials. Such certificates converted into pseudo credentials essentially would look something like:

```
authorizer: "x509-base64:\n  
MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgI2\  
CzA This is would be the public key/certificate of the AQEB\  
BQA signer of the user certificate (i.e. the CA certificate)IUuz\  
eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\  
ct+016zUBP/cQMml+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjPui3AgMB\  
AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\  
iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaa\  
NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\  
yiXHSU8="\  
licensees: "x509-base64:\n  
MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\  
BAMTCUllRUxBQIBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\  
CzA          This is would be the key of the subject of the AQEB\  
BQA          certificate          IUuz\  
eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\  
ct+016zUBP/cQMml+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjPui3AgMB\  
AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\  
iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaa\  
NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\  
yiXHSU8="
```

```

NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\
yiXHSU8="
conditions: app_domain == "IPsec policy" -> "true";

```

Now note that these are not authorized by "POLICY" and thus won't have any effect without a policy authorizing them somehow. Further, this showed what happens to certificates internally. The above credential is thus not seen in the policy file. However, it is possible to sublicense to such credentials. Remember sublicensing above. It is thus possible to license all certificates that are signed by a certain CA by putting the CA certificate as a licensee to "POLICY":

```

keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "x509-base64:\
    MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
    BAMTCUllLRUxBQibDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\
    Cza          This would be the CA certificate          AQEB\
    BQA          IUuz\
    eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
    ct+016zUBP/cQMml+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
    AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\
    iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaa\
    NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\
    yiXHSU8="
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";

```

Thus the above policy is a simple example of a policy that delegates to a CA. Thus any user that has a certificate that is signed by the CA that has this certificate and otherwise comply to other conditions set by the policy and the configuration file would be authorized.

Almost secure...is not secure!

Now, to be really safe, this is not enough unfortunately. There are ways to attack a security gateway that is configured in this way. If you really don't want the details, skip a to the next section now. To everyone else, let's try to understand why this is not secure. It's not hard.

Consider what information one of the isakmpds has access to at this stage. From the configuration file, isakmpd knows which IP-address its peer will send from (in the [Phase 1] section). From the information it gets at the phase 1 negotiation it knows the ID that the peer presents itself with and the certificate it gets from the peer proves that the peer really have this ID. Looks fine so far?

Well, if the ID information was the IP (the default situation if we do not provide a phase 1 ID section) everything would be fine. The CA would have tied the IP to the cert and the IP in the configuration would be all information we need. It would be possible for an imposter to use the same IP from another computer in some cases (i.e. if both computers were on the same local network and the computer that usually have this IP is down for some reason). It should however not be possible for an imposter to be able to have a certificate and a corresponding private key that (falsely) proves that this IP belongs to the imposter.

If that ever happened, the imposter would have managed to either steal the private key from the real owner of the IP, or the imposter would have managed to fool the CA into issuing a certificate containing false information somehow. If any of these things happened, then either the private key had not been protected well enough, the CA had failed to check the identity of the imposter well enough (or the ID info for the cert) or the CA private key had not been well enough protected. Since all these are prerequisites for security to work at all, none of these situations can be allowed to ever occur.

Now, in our example the situation is different. Here we actually have an FQDN in the certificate instead of an IP address. Since we still have an IP address in the [Phase 1] section this will result in a possible security problem. What now would happen during an ISAKMP phase 1 negotiation would be that we could check that the peer was sending from the expected IP (but as explained earlier that could possibly be forged in some situations). We could check that the ID our peer presents actually belongs to our

peer. But what we can not check now is if that ID really is the ID we expect our peer to have, because isakmpd have never been told what ID that should be.

Someone now might say that the DNS system ties the IP to the FQDN for the host. That is true, however today's DNS system is not secure and can, under some circumstances, be fooled to give out false information (or, it could be subject to a denial of service (DoS) attack by an attacker, and the attacker's computer might be able to fake the DNS server's answer). Secure DNS will come in the future, but it is not here yet (at least most DNS servers are not secure yet), thus today, using DNS to check if the FQDN in the cert corresponds to the expected IP is no guarantee. In fact isakmpd does not check this with DNS. Even if DNS was secure, checking this would not help in the case of using an UFQDN.

Thus in the case of having an FQDN as ID, it could be possible for an attacker to get an own private key and having this key signed by the same CA that we use (but with the attacker's own FQDN, of course). Then launch a DoS attack on our peer so that it goes down (in fact, there are some flaws in the ISAKMP protocol itself that possibly could be used to launch a remote DoS against the peer and make it go down, although I don't know how sensitive isakmpd is to those attacks). Then the attacker could configure its own computer in the same way as our peer, connect it to our peers network and try to connect using its own ID, private key and certificate.

Since our own isakmpd has not been informed about what ID to our peer (and it is because the attacker is identically configured as our peer besides the certificate, ID and private key). Further our isakmpd can check that the certificate was signed by the same CA (but most CAs sign lots of certs, a cert might not be hard to get), and that the presented ID is the same as the ID in the cert. However it would not, with the configuration presented so far, check that this ID is the expected ID. Thus the attacker would be allowed to connect.

Preventing the attack.

The question now thus is, how can we inform isakmpd about what ID to expect? This is fortunately easy, and documented in the [isakmpd.policy\(5\)](#). We must do the check in the policy. Like this:

```
keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "x509-base64:\
    MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
    BAMTCUllRUxBQiBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEX\
    Cza          This would be the CA certificate          AQEB\
    BQA          IUuz\
    eOW8P5UGJUH2JVkiaA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
    ct+016zUBP/cQMml+KownxAUq9eza8GvTyUWC97SOMOgoVj/QR3FHmEjPui3AgMB\
    AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMPlad+\
    iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaa\
    NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYZheOxvrtUErRwZ9Xs1KzHe\
    yiXHSU8="
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            remote_id == "gw.worksite.se" -> "true";
```

Now only gw.worksite.se should be able to get an IPsec connection. More allowed IDs could easily be added by adding more alternative remote_id checks, i.e. by having conditions like these in the policy:

```
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            (remote_id == "gw.worksite.se" ||
             remote_id == "gw.whatsite.se") -> "true";
```

With this policy either of gw.worksite.se, gw.somesite.se or gw.whatsite.se could connect.

Some might say that is unfortunate that there has to be entire certificates inserted in the policy. It requires some work to reformat the certificates into the format in the policy and it makes the policy rather unreadable. If someone by mistake replaced a user

certificate with the corresponding CA certificate somewhere in a complex policy it might cause unauthorized users to be allowed to connect in some cases and worse, it would not be easily detectable by reading through the policy file (since X.509 certificates are not in a human readable format).

Now, for the really bleeding edge people out there a solution to this problem is available. It is now possible to use the certificate Distinguished Name (DN) instead of a certificate in the policy (the corresponding certificate must of course be available from the certs or ca directories on disk so that isakmpd can find it). With this format the policy above might look like something like this instead:

```
keynote-version: 2
comment: This is an example of a policy delegating to a key.
authorizer: "POLICY"
licensees: "DN:\C=se\CN=IKELAB CA"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" &&
            (remote_id == "gw.worksite.se" ||
             remote_id == "gw.somesite.se" ||
             remote_id == "gw.whatsite.se") -> "true";
```

Much more readable, isn't it? The information about what the exact DN for a certificate is can be found by looking at the certificate using the openssl utility. Something like:

```
$ openssl x509 -text < ca.crt
```

More complicated policy configurations are of course possible but this is a start anyway, and there are another example in the next section.

This should provide the necessary information about the certificate in ca.crt. Now this is good as long as we have a small or at least reasonably small policy. It is however still not too great if we have a gigantic site with lots of users that should be allowed to connect.

Multiuser configurations and/or centrally managed authorisation.

Now, lets look at some really cool features of isakmpd. Previously we assumed that the expected peer was well known and had a static IP address. This is not always the case. Lots of people use dynamically assigned IPs or use many different computers. In other cases (like for a server) we might not know for sure who wants to connect.

Therefore one very nice feature of isakmpd is the ability to use a default tag instead of an IP in the [Phase 1] section, thus allowing isakmpd negotiations from any IP. This might thus look something like this:

```
[phase 1]
Default=          work-gw
```

First, it should be said that this configuration might not be secure from DoS attacks. As said before, there are some flaws in the ISAKMP/IKE protocols. Anyway, using a default [phase 1] section also enable us to use a kind of "authorisation certificates" instead.

Consider the case where we have a lot of authorized users but when we would not accept just any user. Like at a company. We would like company employees to be allowed to connect but nobody else. Now, imagine a big company where there might be thousands of employees. We might like them to all be able to connect from any computer (not only from within the company network) but everyone should not be allowed to do anything. It should now be possible to write a policy like this:

```
keynote-version: 2
authorizer: "POLICY"
licensees: "telnet@work" || "telnet@lab" || "pop3@work"
conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
```

```

esp_enc_alg != "null" &&
remote_id_type == "UFQDN" &&
(remote_id == "telnet@worksite.se" ||
 remote_id == "pop3@worksite.se" ||
 remote_id == "telnet@lab.worksite.se") -> "true";

```

```

authorizer: "telnet@work"
licensees: "DN:\C=se\CN=IKELAB CA"
conditions: remote_id == "telnet@worksite.se" &&
             local_filter_type == "IPv4 address" &&
             local_filter_port == "23" &&
             local_filter == "192.168.002.003"

```

```

authorizer: "telnet@lab"
licensees: "DN:\C=se\CN=IKELAB CA"
conditions: remote_id == "telnet@lab.worksite.se" &&
             local_filter_type == "IPv4 address" &&
             local_filter_port == "23" &&
             local_filter == "192.168.002.002" -> "true";

```

```

authorizer: "pop3@work"
licensees: "DN:\C=se\CN=IKELAB CA"
conditions: local_filter_type == "IPv4 address" &&
             local_filter_port == "110" &&
             local_filter == "192.168.002.003" &&
             remote_id == "telnet@worksite.se" -> "true";

```

This might not be exactly how it should be. This is as far as I know completely untested (in fact, these filter conditions might not work at all as I expect). Also, a policy such as this one (in fact any with default as peer IP), would require rewrites of the `isakmpd.conf` file too. This would have some security implications too. Further, for this kind of connections where anyone should be allowed to connect, it would probably be desirable to log the DN of anyone who connected. `Isakmpd` does not yet support that to my knowledge. Also this probably could have other security implications. You are on your own, you have been warned. The basic idea should be clear anyway.

Just in case someone missed the really interesting possibilities this would have. If all computers using ISAKMP/IKE this way had a standard set of conditions for all services the users might like to use from remote, the CA could actually authorize users by just putting the right `SubjectAltName` extensions in their certificates. Further, the expiration time for such certificates could be set to expire relatively often although the users would be able to download new reissued certificates when their current certificate is getting old. If the users misuse their authorizations, just stop reissuing the certificates and they won't get in more after it has expired. No need to change policy files on all computers just because an employee i.e. quits their job. The same scheme should work for other purposes than ISAKMP/IPsec too (including authorization for off-line systems!) although that would require special software. In any case, any organisation doing this would probably want to be their own CA.

Multiuser configurations (mobile users) like these are possible with pre-shared keys too, but then it is required that `AGGRESSIVE` mode is used instead of `ID_PROT` mode since we then must be able to choose the right password phrase based on ID since we do not know that from what the IP is in this case (in `AGGRESSIVE` mode the ID is sent over at an earlier stage of the negotiation, but it is sent unencrypted, thus `AGGRESSIVE` mode is faster because it needs fewer message exchanges, but it also is a bit less secure since the ID is sent in clear).

13.10 - What IKE clients are compatible with isakmpd?

`isakmpd` is the ISAKMP/Oakley key management daemon that comes with OpenBSD. We suspect that it interoperates, at least partially, with most ISAKMP implementations, but the following have actually been tested. Note that some isakmp software out there are actually based on the OpenBSD isakmp daemon.

The following MS-Windows clients have been reported to be compatible:

- [TimeStep](#) PERMIT/Client
- [Ashley Laurent](#) VPCOM VPN software
- [PGP VPN](#) software
- [Radguard](#) cIPRO client
- [Cisco](#) IRE client
- [Microsoft](#) Windows 2000 (Transport mode only)

The following gateways/routers have been reported to be compatible:

- [Cisco](#) IOS
- [Cisco](#) PIX
- [Intel](#) LanRover
- [TimeStep](#) PERMIT/Gate
- [Cendio](#) Fuego
- [KAME](#) for FreeBSD
- [FreeS/WAN](#) for Linux
- [Axent](#) Raptor
- [Ericsson](#) eBox
- [Radguard](#) cIPRO-VPN
- [F-Secure](#) VPN+
- [Teamware](#) TWISS
- [3com](#) Pathbuilder
- [Nortel](#) Contivity
- [CheckPoint](#) FW-1

13.11 - Troubleshooting IPsec/VPN

Your first tool for troubleshooting IPsec is [tcpdump\(8\)](#). Use `tcpdump` to look for several things.

First, if you are using `tcpdump` from OpenBSD, you have an enhanced version of `tcpdump` which can show some information about ESP and AH packets. If you are using `tcpdump` from OpenBSD 2.5 or on another operating system, chances are you have an older version that will simply show the protocol number for AH or ESP. (ESP is IP protocol 50, AH is 51)

- With `tcpdump`, look and see if traffic is using AH/ESP or cleartext. If your traffic is in cleartext, then your flows are setup incorrectly or your isakmp is not negotiating properly. Use [ping\(8\)](#) to generate simple traffic.

For instance, I have two hosts, 208.1.1.1 and 208.2.2.2. Logged in to 208.2.2.2, I am doing this:

```
vpn# ping -c 3 208.1.1.1
PING esp.mil (208.1.1.1): 56 data bytes
64 bytes from 208.1.1.1: icmp_seq=0 ttl=255 time=190.155 ms
```

```
64 bytes from 208.1.1.1: icmp_seq=1 ttl=255 time=201.040 ms
64 bytes from 208.1.1.1: icmp_seq=2 ttl=255 time=165.481 ms
--- esp.mil ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 165.481/185.558/201.040 ms
```

And in another session, I can see my encapsulated pings:

```
vpn# tcpdump -ni fxp7 host 208.1.1.1
tcpdump: listening on fxp7
14:12:19.630274 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4535 len 116
14:12:19.813519 esp 208.1.1.1 > 208.2.2.2 spi 0x00001001 seq 49313 len 116
14:12:20.630277 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4536 len 116
14:12:20.832458 esp 208.1.1.1 > 208.2.2.2 spi 0x00001001 seq 49314 len 116
14:12:21.630273 esp 208.2.2.2 > 208.1.1.1 spi 0x00001000 seq 4537 len 116
^C
1831 packets received by filter
0 packets dropped by kernel
```

- ISAKMP runs on UDP port 500. If this is locked out through a firewall or packet filter, then you need to change it!

```
# Passing in ISAKMP traffic from the security gateways
pass in on ne0 proto udp from gatewB/32 port = 500 to gatewA/32 port = 500
pass out on ne0 proto udp from gatewA/32 port = 500 to gatewB/32 port = 500
```

```
# Passing in encrypted traffic from security gateways
pass in proto esp from gatewB/32 to gatewA/32
pass out proto esp from gatewA/32 to gatewB/32
```

- Photuris runs on UDP port 468. Same considerations, but the protocol is ah.

```
# Passing in Photuris traffic from the security gateways
pass in on ne0 proto udp from gatewB/32 port = 468 to gatewA/32 port = 468
pass out on ne0 proto udp from gatewA/32 port = 468 to gatewB/32 port = 468
```

```
# Passing in encrypted traffic from security gateways
pass in proto ah from gatewB/32 to gatewA/32
pass out proto ah from gatewA/32 to gatewB/32
```

- To turn on all debugging in isakmpd, start it as

```
# /sbin/isakmpd -d -DA=99
```

or (to skip the most detailed timer debug info)

```
# /sbin/isakmpd -d -DA=99 -D1=70
```

- You need to allow traffic which has been processed by IPsec from netB into your local firewalled netA.

```
# Passing in traffic from the designated subnets.
pass in on enc0 from netB/netBmask to netA/netAmask
```

- With tcpdump on OpenBSD, you can decode most cleartext parts of Internet Key Exchange sessions. Tcpcmdump will also show AH payload data.
- Mount a /kern filesystem (if you don't use one by default already.)

```
# mkdir /kern; mount -t kernfs /kern /kern
```

In /kern, there is a table of current SA/SPIs, including which have flows (outgoing SAs) or not (incoming SAs). There are also traffic counters which you can use to see what traffic is going where.

- Finally, you can use [netstat\(1\)](#) to see your SAs.

```
vpn% netstat -rn -f encap
Routing tables
```

```
Encap:
Source          Port  Destination          Port  Proto SA(Address/SPI/Proto)
```

0.0.0.0/32	0	192.168.99/24	0	0	208.1.1.1/00001000/50
0.0.0.0/32	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.1.2.0/24	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.1.2.0/24	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.1.5.0/24	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.1.5.0/24	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50
208.2.2.2/32	0	192.168.99/24	0	0	208.1.1.1/00001000/50
208.2.2.2/32	0	208.1.1.1/32	0	0	208.1.1.1/00001000/50

- If all else fails, recompile your kernel with option `ENCDEBUG`. Then, set the `sysctl net.inet.ip.encdebug` to 1. Look in your `dmesg` for warnings or errors, and report them using [sendbug\(1\)](#) to the OpenBSD developers. Alternately, if you are not sure you have actually run into a bug, you may want to send a message to one of the [mailing lists](#).

13.12 - Related Documentation

IPsec is partially documented in the [vpn\(8\)](#) man page. There are various configuration templates in `/usr/share/ipsec/` directory which can also assist you. The manual pages for [enc\(4\)](#), [ipsec\(4\)](#), [ipsecadm\(8\)](#), [photurisd\(8\)](#), [startkey\(1\)](#), [isakmpd\(8\)](#), [isakmpd.conf\(5\)](#) and [isakmpd.policy\(5\)](#) are detailed and can assist in setup and operation of IPsec.

Other links...

- [IETF IPsec Working Group](#)
- [SSH IPsec interoperability Test Node](#)
- [NIST IPsec Web Based Interoperability Tester](#)
- [A port of OpenBSD's IPsec to FreeBSD](#)
- [FreeS/WAN - IPsec for Linux](#)
- [Lots of IPsec related links](#)
- [OpenBSD 2.4 VPN Configuration Mini-FAQ](#)

And on to the RFCs...

- [RFC 1320](#) - The MD4 Message-Digest Algorithm
- [RFC 1321](#) - The MD5 Message-Digest Algorithm
- [RFC 1828](#) - IP Authentication using Keyed MD5
- [RFC 1829](#) - The ESP DES-CBC Transform
- [RFC 2040](#) - The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms
- [RFC 2085](#) - HMAC-MD5 IP Authentication with Replay Prevention
- [RFC 2104](#) - HMAC: Keyed-Hashing for Message Authentication
- [RFC 2144](#) - The CAST-128 Encryption Algorithm
- [RFC 2202](#) - Test Cases for HMAC-MD5 and HMAC-SHA-1
- [RFC 2207](#) - RSVP Extensions for IPsec Data Flows
- [RFC 2268](#) - A Description of the RC2 Encryption Algorithm
- [RFC 2367](#) - PF_KEY Key Management API, Version 2
- [RFC 2401](#) - Security Architecture for the Internet Protocol (**IPsec**)
- [RFC 2402](#) - IP Authentication Header (**AH**)
- [RFC 2403](#) - The Use of HMAC-MD5-96 within ESP and AH
- [RFC 2404](#) - The Use of HMAC-SHA-1-96 within ESP and AH
- [RFC 2405](#) - The ESP DES-CBC Cipher Algorithm With Explicit IV

13.0 - Using IPsec

- [RFC 2406](#) - IP Encapsulating Security Payload (**ESP**)
- [RFC 2407](#) - The Internet IP Security Domain of Interpretation for ISAKMP
- [RFC 2408](#) - Internet Security Association and Key Management Protocol (**ISAKMP**)
- [RFC 2409](#) - The Internet Key Exchange (**IKE**)
- [RFC 2410](#) - The NULL Encryption Algorithm and Its Use With IPsec (ha ha...)
- [RFC 2411](#) - IP Security Document Roadmap
- [RFC 2412](#) - The OAKLEY Key Determination Protocol
- [RFC 2451](#) - The ESP CBC-Mode Cipher Algorithms
- [RFC 2522](#) - Photuris: Session-Key Management Protocol
- [RFC 2523](#) - Photuris: Extended Schemes and Attributes
- [RFC 2631](#) - Diffie-Hellman Key Agreement Method
- [RFC 2709](#) - Security Model with Tunnel-mode IPsec for NAT Domains

[\[Back to Main Index\]](#) [\[To Section 12.0 - For Advanced Users\]](#) [\[To Section 14.0 - Using disks in OpenBSD\]](#)



www@openbsd.org

\$OpenBSD: faq13.html,v 1.34 2000/05/26 07:10:32 chris Exp \$

